다중처리기 시스템을 위한 적응적 작업할당 방법의 개선

옥 기 상† · 박 준 석†† · 이 원 주††† · 전 창 호††††

요 약

다중처리기 시스템을 위한 적응적 작업할당 방법에서는 작업의 대기시간을 줄이기 위하여 사용 가능한 서브큐브에 맞도록 작업의 크기를 반으로 줄인다. 이 방법에서는 작업을 수행하기에 적합한 서브큐브가 존재하지 않을 경우 작업크기를 축소하여 할당함으로써 대기시간을 줄일 수 있지만, 작업크기 축소로 인해 늘어나게 되는 작업수행시간 때문에 전체작업수행비용이 오히려 증가될 수도 있다.

본 논문에서는 기존의 적응적 작업할당 방법을 개선한 Estimate-fold 작업할당 방법을 제안한다. 이 방법에서는 작업을 수행하기에 적합한 서브큐브가 존재하지 않으면 적합한 서브큐브가 형성될 때까지 대기하는 경우와 작업크기를 축소하여 할당하는 경우에 대한 작업수행비용을 예측하여 그 중에서 유리한 쪽을 선택한다. 시뮬레이션으로 Estimate-fold 작업할당 방법과 기존의 적응적 작업할당 방법, 그리고 버디, 그레이코드 작업할당 방법들의 평균작업수행비용을 구하여 비교 평가함으로써 제안하는 Estimate-fold 작업할당 방법이 가장 효율적임을 보인다.

An Improved Adaptive Job Allocation Method for Multiprocessor Systems

Ki-Sang Ok† · Joon-Seok Park†† · Won-Joo Lee††† · Chang-Ho Jeon††††

ABSTRACT

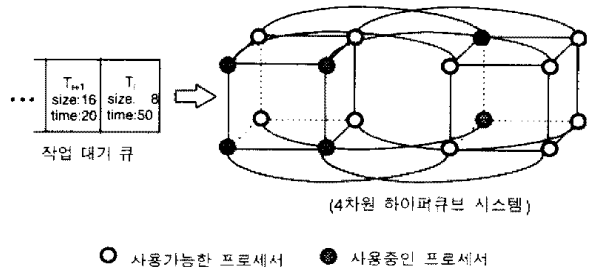
In adaptive job allocation method for multiprocessor systems a job is folded, or split in two halves, to fit for an available subcube in order to reduce the waiting time of jobs. In this method, however, since a job is folded whenever a subcube with the proper size is not found, the prolonged execution time caused by job split may override the savings in waiting time, in which case the total execution cost of jobs may be increased.

In this paper, an improved adaptive job allocation algorithm, called Estimate-fold allocation, is presented and evaluated. The proposed algorithm estimates the costs and takes the better of two alternatives; folding right away and waiting until a bigger subcube becomes available. The average total job execution cost of our algorithm is calculated and compared to those of the conventional adaptive, buddy, and gray-code algorithms through simulations. The results shows that our proposed algorithm performs better than others.

※ 이 논문은 1997년 한양대학교 교내연구비에 의하여 연구되었음.
† 정 회 원 : 한국통신 연구개발본부 통신망연구소
†† 준 회 원 : 한양대학교 대학원 전자계산학과
††† 준 회 원 : 두원공과대학 소프트웨어개발과 교수
†††† 종신회원 : 한양대학교 전자컴퓨터공학부 교수
논문접수 : 1999년 1월 6일, 심사완료 : 1999년 5월 13일

1. 서 론

다중처리기 시스템에서는 프로세서들간의 통신비용이 최소가 되도록 서로 인접한 프로세서들로 형성된 서브큐브에 작업을 할당하는 것이 원칙이다. 그래서 어떤 작업을 수행하는데 필요한 것 보다 많은 프로세서가 시스템에 존재한다 하더라도 작업을 수행하기에 적합한 서브큐브가 구성되지 않으면 작업을 할당 할 수 없기 때문에 프로세서 단편화(processor fragmentation)가 발생한다[1-4]. 4차원 하이퍼큐브 시스템에서 큐에 대기중인 작업(T_i, T_{i+1},)들을 할당하는 (그림 1)을 예로 들어 프로세서 단편화 현상을 설명하면 다음과 같다.



(그림 1) 프로세서 단편화의 예

(그림 1)에서 크기 8인 작업 T_i를 할당하려면 8개의 유휴 프로세서로 구성된 3차원 서브큐브가 존재해야 한다. 그러나 다른 작업을 수행중인 6개의 프로세서와 10개의 유휴 프로세서가 존재하는 4차원 하이퍼큐브에는 작업 T_i를 수행하기에 적합한 서브큐브가 형성되어 있지 않다. 이 때 작업 T_i는 큐에 대기하게 되고 10개의 유휴 프로세서를 사용하지 못하는 프로세서 단편화 현상이 발생한다.

프로세서 단편화 현상을 줄이기 위해서는 시스템에서 사용 가능한 프로세서를 찾아내는 프로세서 인식율을 높여야 한다. 프로세서 인식율을 높여 프로세서들을 최대한 활용 할 수 있는 여러 작업할당 방법들이 제안되었다. 작업할당 방법은 프로세서 인식율을 높이는 방법에 따라 BMT(bit-mapped technique)와 ACT(available cube technique)로 분류된다[5]. BMT는 프로세서 수 만큼의 비트를 가지고 각 프로세서의 사용여부를 표현하여 서브큐브 인식율을 높이는 방법으로 버디(buddy)[6], 변형 버디(modified buddy)[7], 그레이코드(gray code)[8], 다중 그레이코드(multiple gray code)[8] 할당방법 등이 이에 속한다. 이 방법은 단순하지만 서브큐브 인식능력이 ACT방법에 비해 낮다. ACT는 사용 가능한 프로세서들을 적합한 큐브 형태로 표현하여 서브큐브 인식율을 높이는 방법으로 free list[7], MSS(maximal subset of subcubes)[9], PC(prime cube) graph[10] 할당방법 등이 있다. ACT방법을 사용하는 할당 정책들은 대부분 완벽한 서브큐브 인식율을 가지지만 할당이나 해제 후에도 계속해서 시스템을 조건에 맞게 일관된 형태를 유지해야 하기 때문에 할당과 해제 알고리즘이 복잡하다.

버디 할당방법은 n차원 시스템에 크기 2^k인 작업을 할당 할 때 서브큐브의 영역내의 모든 프로세서가 사용 가능하면 작업을 할당하며, 단순하지만 사용 가능한 모든 프로세서를 인식하지 못한다는 단점이 있다[7][9]. 버디 할당방법에 작업크기 축소의 개념을 적용한 것이 적응적 작업할당 방법이다[1]. 버디 작업할당 방법에서는 시스템에 작업을 수행하기에 필요한 프로세서 수 이상의 유휴 프로세서가 존재하더라도 적합한 서브큐브가 형성되어 있지 않으면 서브큐브가 형성될 때까지 큐에서 대기하게 된다. 그러나 적응적 작업할당 방법에서는 작업을 수행하기에 적합한 서브큐브가 형성되어 있지 않을 경우 대기시간을 줄이기 위해 무조건 작업크기를 축소하여 할당한다. 이 때 작업크기를 축소로 인해 늘어나는 작업수행시간이 줄어든 대기시간보다 커질 수도 있으며 이 경우 전체작업수행비용(the total execution cost of jobs)이 오히려 증가하게 되는 문제점이 있다.

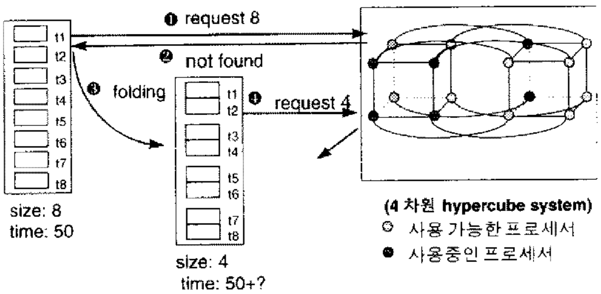
본 논문에서는 적응적 작업할당 방법의 문제점을 개선한 Estimate-fold 작업할당 방법을 제안한다. Estimate-fold 작업할당 방법에서는 작업을 수행하기에 적합한 서브큐브가 존재하지 않으면 먼저 적합한 서브큐브가 형성될 때까지 기다리는 경우의 작업수행비용과 작업크기를 축소하여 할당하는 경우의 작업수행비용을 구하여 그 중에서 최소 작업수행비용을 갖는 할당을 선택하게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 적응적 작업할당 방법과 문제점에 대하여 설명한다. 그리고 3장에서는 본 논문에서 제안하는 Estimate-fold 작업할당 방법에 대하여 자세히 설명한다. 4장에서는 제안된 작업할당 방법의 효율성을 입증하기 위한 시뮬레이션 환경을 설명하고 시뮬레이션 결과를 비교 분석하며, 끝으로 5장에서 결론을 맺는다.

2. 적응적 작업할당 방법

적응적 작업할당 방법은 작업을 할당하기에 적합한 서브큐브가 없으면 작업크기를 축소하여 할당을 시도하는 방법으로 ***RSR t***(restricted size reduction) 할당 방법이라고도 한다. 여기서 t는 폴딩 횟수를 의미하고 폴딩(folding)은 작업크기를 1/2로 축소하는 과정을 의미한다. 폴딩 횟수 t는 작업크기에 따라 달라지며 작업크기가 2^k일 때 최대 폴딩 횟수 t는 k 1이 된다[1].

6개의 사용중인 프로세서와 10개의 유휴 프로세서로 구성된 4차원 하이퍼큐브 시스템에 크기가 8이고 수행시간이 50인 작업을 할당하는 (그림 2)를 예로 들어 적응적 작업할당 과정을 설명한다.



(그림 2) 적응적 작업할당 과정

① 크기가 8인 작업은 수행하기에 적합한 서브큐브를 4차원 하이퍼큐브에 요청한다.
② 4차원 하이퍼큐브는 10개의 유휴 프로세서를 가지고 있지만 작업크기에 적합한 서브큐브가 형성되어 있지 않음을 알린다. 작업크기에 적합한 서브큐브를 찾는 것은 프로세서들간의 통신비용을 최소화하기 위함이다.
③ 대기시간을 줄이기 위해 폴딩으로 작업크기를 1/2로 축소한다. 폴딩 과정은 작업크기가 8인 작업의 부작업을 2개씩 묶는 클러스터링(clustering)으로 작업크기를 4로 축소한다. 이때 축소된 작업의 수행시간은 증가하게 된다.
④ 작업크기가 4로 축소된 작업은 수행에 적합한 서브큐브를 4차원 하이퍼큐브에 재요청한다. 이때 적합한 서브큐브가 존재하면 작업을 할당하고 서브큐브가 존재하지 않으면 과정 ③~④를 반복하며 작업 할당을 시도한다.

(그림 2)의 적응적 작업할당 과정을 간략화 된 코드 형태로 표현하면 (그림 3)과 같다.

```
[2^k 서브큐브을 요구하는 작업 Jm을 할당할 경우]
1. s = MIN(k-t, 0)  //s는 최소 작업크기,
                     t는 최대 폴딩 가능 횟수
2. if (2^k 서브큐브가 존재하는가?){
        2^k 서브큐브에 작업 Jm을 할당.
        Goto step-3
  }else{
        k := k  1     //작업크기 축소
        if (k  s){
                goto step 2.
        }else
                goto step-4
        }
  }// end of if
3. If (큐가 empty가 아닌가?){
         queue의 첫 작업에 대해서, goto step-1.
  }
4. End
```

(그림 3) 적응적 작업할당 알고리즘

(그림 3)의 적응적 작업할당알고리즘에서는 크기가 2^k인 작업 J_m을 할당한다. 먼저 과정 1에서는 최소 작업크기를 구한다. 과정 2에서는 작업 J_m을 수행하기에 적합한 서브큐브를 요청한다. 이때 적합한 서브큐브가 형성되어 있으면 작업 J_m을 할당한다. 그러나 서브큐브가 형성되어 있지 않으면 적합한 서브큐브가 형성될 때까지 기다리는 대기시간을 줄이기 위해 작업크기를 2^{k-1}로 축소하여 할당한다. 과정 3에서는 큐에 할당 대기중인 작업이 존재하는가를 확인하고, 작업이 할당 대기중이면 과정 1~3를 반복한다.

적응적 작업할당알고리즘의 과정 2에서 작업 J_m을 수행하기에 적합한 서브큐브를 찾지 못하면폴딩으로 작업크기를 2^{k-1}로 축소하여 할당함으로써 대기시간을 줄인다. 이 때 줄어는 대기시간보다 폴딩으로 늘어난 수행시간이 커질 수 있으며 이럴 경우 전체작업수행비용이 증가하여 오히려 시스템의 성능을 저하시키는 문제점이 있다.

3. Estimate-fold 작업할당 방법

본 논문에서는 앞에서 설명한 적응적 작업할당 방법의 문제점을 개선한 Estimate-fold 작업할당 방법을 제안한다. Estimate-fold 작업할당 방법은 작업을 수행하기에 적합한 서브큐브가 형성되어 있지 않을 경우 적합한 서브큐브가 형성될 때까지 대기하는 경우의 작

업수행비용과 최대 폴딩 횟수 1회가 반복되면서 작업 크기를 축소하여 할당하는 경우의 작업수행비용을 구한다. 그리고 이렇게 구한 작업수행비용들을 비교하여 최소 작업수행비용을 갖는 작업할당을 선택한다.

Estimate-fold 작업할당 방법은 기존의 적응적 작업할당 방법에 다음과 같은 사항을 추가적으로 고려한다.

첫째, 폴딩으로 인해 증가하는 작업의 수행시간을 고려한다. 이 수행시간은 시스템의 토폴로지, 상호연결망의 대역폭, 그리고 프로토콜과 동일한 작업이 할당된 프로세서간의 통신 빈도수에 따라 증가율이 달라지게 된다. 먼저 하이퍼큐브의 경우 1개의 프로세서에 비해 n개의 프로세서에서 수행될 때 속도 향상을 계산하는 식은 다음과 같다[11].

$$S(n) = \frac{T_1}{T_n} = \log n \qquad (1)$$

식 (1)에서 T_1은 1개의 프로세서에서 수행되는 수행시간을 의미하고, T_n은 n개의 프로세서에서 수행되는 수행시간을 의미한다. 프로세서의 수가 4 이하일 경우 속도 향상은 선형적으로 이루어 진다[11]. 그러므로 작업크기(N)가 4 이하인 작업을 폴딩 할 때 증가하는 수행시간($T_{N/2}$)을 구하는 식은 다음과 같다.

$$T_{N/2} = 2 \times T_N \quad [\text{작업 크기}(N) \le 4] \qquad (2)$$

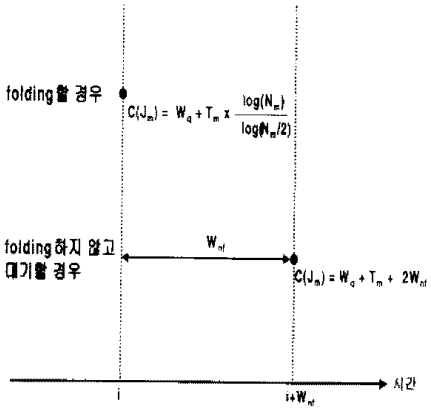
그러나 작업크기(N)가 4 이상일 때 작업의 수행시간(T_N)은 식 (1)에서 변형된 $T_n = T_1/\log n$에 크기 N을 대입한 식 $T_N = T_1/\log N$으로 구한다. 그리고 크기 N인 작업을 폴딩하면 그 크기가 N/2로 축소되기 때문에 수행시간($T_{N/2}$)을 구하는 식은 $T_N = T_1/\log N$ 식에 N/2을 대입하여 다음과 같이 구한다.

$$T_{N/2} = T_1/\log(N/2)$$
$$= (T_N \times \log N)/\log(N/2) \ [\text{작업 크기}(N) > 4] \qquad (3)$$

식 (3)의 T_1은 식 (1)에서 $T_N \times \log N$으로 구할 수 있다.

둘째, 수행하기에 적합한 서브큐브가 형성될 때까지 폴딩하지 않고 기다리는 대기시간(W_{nf})과 그 만큼 할당이 늦어지는 후속 작업의 할당 지연시간을 고려한다.

(그림 4)에서 작업 J_m의 수행시간은 T_m이고, 작업크기는 N_m이다. W_q는 작업 J_m이 큐에서 대기한 시간이다. 그리고 W_{nf} 은 작업 J_m을 폴딩하여 할당하지 않고 수행하기에 적합한 서브큐브가 형성될 때까지 대기하는 시간을 의미한다. 할당 대기중인 작업 J_m은 임의의 한 시점 i에서 할당할 때 시스템에 적합한 서브큐브가 존재하지 않으면 폴딩하여 할당한다. 이 때 폴딩으로 인해 증가하는 작업의 수행시간은 $T_m \times ((\log(N_m)/\log(N_m/2))$이 되고 대기시간($W_{nf}$)은 0가 된다. 따라서 작업 J_m의 작업수행비용은 폴딩으로 증가한 작업의 수행시간과 큐에서의 대기시간(W_q)을 합산하여 $W_q + T_m \times \log((N_m)/\log(N_m/2))$이 된다. 그러나 작업 J_m을 폴딩하지 않고 대기시간(W_{nf})만큼 대기한 후 할당할 경우 작업 J_m의 작업수행비용은 큐에서의 대기시간(W_q)과 적합한 서브큐브가 형성되기를 기다리는 대기시간(W_{nf}) 그리고 작업 수행시간(T_m)을 합산하여 $W_q + W_{nf} + T_m$이 된다.



(그림 4) 작업(J_m)의 대기시간

이 때 후속 작업 J_{m+1}은 작업 J_m의 대기시간(W_{nf})만큼 할당이 지연되며, 작업 J_m의 대기시간(W_{nf})과 작업 J_{m+1}의 할당 지연시간(W_{nf})이 같은 크기로 동시에 발생한다. 그러므로 작업 J_m의 수행비용 $C(J_m)$은 작업 J_m의 대기시간(W_{nf})과 작업 J_{m-1}의 할당 지연시간(W_{nf})을 고려하여 다음과 같이 구한다.

$$C(J_m) = W_q = T_m + W_{nf} + W_{nf} = W_q + T_m + 2W_{nf} \qquad (4)$$

셋째, 폴딩으로 인해 증가하는 수행시간(T_m)과 폴딩하여 할당하지 않고 적합한 서브큐브가 형성될 때까지 기다리는 대기시간(W_{nf})을 비교하여 작업의 폴딩 여부를 결정한다. (그림 4)를 예로 들어 할당 대기중인 작업 J_m을 폴딩하여 할당할 경우와 폴딩하지 않고 대기시간(W_{nf})만큼 대기한 후 할당할 경우에 대하여 작업

J_m의 수행비용 $C(J_m)$을 구해보자. 먼저 수행시간(T_m)이 100 단위시간, 크기(N_m)가 16인 작업 J_m이 할당 대기중이고 시스템에는 작업 J_m을 수행하기에 적합한 서브큐브가 없다고 가정한다. 폴딩을 통하여 작업크기(N_m)를 8로 축소하면 작업의 수행시간(T_m)은 식 (3)에 의해 400/3 단위시간으로 증가한다. 한 번 폴딩한 결과 8개의 프로세서로 이루어진 서브큐브가 존재하여 작업을 할당하면 대기시간(W_{mf})은 0가 되고 작업 J_m의 수행비용 $C(J_m)$은 큐에서의 대기시간(W_q)과 증가된 작업수행시간(400/3)을 합산하여 W_q+(400/3)이 된다. 그러나 폴딩하지 않고 10 단위시간을 대기한 후 작업 J_m을 수행하기에 적합한 서브큐브가 형성되어 할당된다면 대기시간(W_{mf})은 10이 된다. 이때 작업 J_m의 수행비용 $C(J_m)$은 식 (4)와 같이 큐에서의 대기시간(W_q)과 수행시간(T_m), 그리고 폴딩하지 않고 대기한 시간(W_{mf})과 후속 작업 J_{m+1}의 할당 지연시간(W_{mf})을 합산하여 120 + W_q이 된다. 이와 같이 대기시간(W_{mf})과 폴딩으로 인해 증가하는 수행시간(T_m)을 고려하지 않고 작업 J_m을 무조건 폴딩하여 할당할 경우 작업 J_m의 수행비용 $C(J_m)$이 폴딩하지 않고 대기하는 경우에 비해 40/3 단위시간 만큼 오히려 증가하는 문제점이 발생한다. 이러한 문제점을 해결하기 위해 본 논문에서는 대기시간(W_f)과 폴딩으로 인해 증가하는 수행시간(T_m)을 비교하여 대기시간(W_{mf})이 수행시간(T_m)보다 클 경우에만 폴딩하여 할당한다.

위의 세가지 사항들을 고려하여 기존의 적응적 작업할당 방법을 확장한 Estimate-fold 작업할당 방법은 작업을 수행하기에 적합한 서브큐브가 존재하지 않으면 무조건 작업크기를 축소하지 않고 적합한 서브큐브가 형성될 때까지 기다리는 경우의 작업수행비용과 작업크기에 따라 폴딩 횟수 t에 이르기까지 단계적으로 폴딩하여 할당하는 과정을 반복하며 작업수행비용을 구한다. 이렇게 구한 작업수행비용들을 비교하여 그 중에서 최소 작업수행비용을 갖는 할당을 선택한다. Estimate-fold 작업할당 알고리즘을 간략화 된 코드로 표현하면 (그림 5)와 같다.

(그림 5)의 Estimate-fold 작업할당 알고리즘에서는 작업크기가 2^k인 작업 J_m을 할당한다. 먼저 과정 1에서 작업 J_m의 최대 폴딩 횟수 t를 구한다. 과정 2에서는 작업 J_m을 수행하기에 적합한 서브큐브를 요청한다. 만약 서브큐브가 존재하면 작업을 할당한다. 그러나 서브큐브가 존재하지 않으면, 먼저 2^k개의 사용 가능한 프로세서로 구성된 서브큐브가 형성될 때까지 기다리는 대기시간(W_{mf})과 수행시간을 고려하여 수행비용을 예측한다. 그리고, 적합한 서브큐브를 찾을 때까지 폴딩을 반복하며 대기시간(W_{mf})과 큐에서의 대기시간(W_q), 폴딩으로 인해 증가된 수행시간을 합산하여 작업수행비용을 예측하는 과정을 최대 t번까지 반복한다. 과정 3에서는 과정3에서 구한 작업수행비용중 최소값을 갖는 작업을 할당한다. 과정 4에서는 큐에 할당 대기중인 작업이 존재하는가를 확인하고, 작업이 할당 대기중이면 과정 1~3를 반복한다.

```
[크기 2^k인 작업 Jm이 할당 대기중일 때]
1. t = k-1;
2. if (할당에 적합한 2^k 서브큐브가 존재하는가?){
     2^k 서브큐브에 작업 Jm을 할당.
     Goto step-4
  }else
     T[1] = 큐에서의 대기시간(Wq) + 수행시간 + 대기시간(Wmf)*2
     For (i = 1; i <= t; i++){
        폴딩으로 작업의 크기를 2^(k-i) 서브큐브에 적합하도록 줄임
        if (할당에 적합한 2^(k-i) 서브큐브가 존재하는가?){
           T[i*2] = 큐에서의 대기시간(Wq) + 폴딩으로 증가한 수행시간
        }else
           T[i*2+1] = 큐에서의 대기시간(Wq) + 수행시간 + 대기시간(Wmf)*2
        }
     }// end of for
  }// end if
3. 각 T[i]값중 가장 작은 값을 가진 방법을 취한다.
4. if (큐가 empty가 아닌가?){
        queue에 첫 작업에 대해서, goto step-1.
  }
5. End.
```

(그림 5) Estimate-fold 작업할당 알고리즘

4. 성능평가

4.1 전제 조건

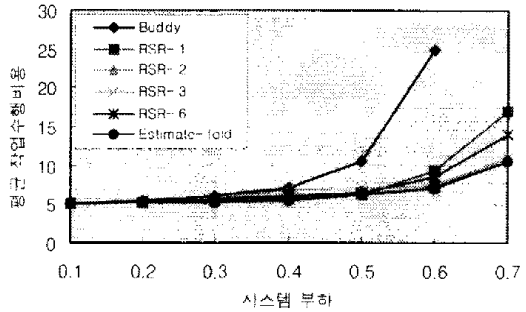
일반적으로 작업들은 전처리 단계인 분할과정에서 독립적으로 실행 가능한 단위로 분할되며, 그 때 각 작업들의 크기와 수행시간, 작업들간의 종속 관계까지 정해진다[12]. 그러나 본 논문에서는 기존의 적응적 작업할당 방법[1]과 같은 조건에서 비교 될 수 있는 기준을 가지기 위하여 작업들간의 전후 순서 관계는 고려하지 않는다. 작업크기는 축소가 용이하고 내적 단편화가 발생하지 않도록 2의 지수 승으로 주어진다고 가정한다. 또한, 단일 큐에 도착하는 작업들은 포아송(Poisson) 분포를 따르며, 작업크기와 수행시간은 정규분포를 따른다고 가정한다. 단일 큐에 대기중인 작업들은 FCFS(first come first service) 스케줄링으로 서비스를 받는다고 가정한다.

4.2 성능 평가

본 논문에서는 하이퍼큐브 시스템을 대상으로 기존의 적응적 작업할당 방법의 성능을 평가한 시뮬레이션 환경과 동일한 조건하에서 버디 할당방법, 그레이코드 할당방법, 그리고 기존의 적응적 작업할당 방법과 제안된 Estimate-fold 작업할당 방법의 성능을 비교 평가함으로써 그 결과에 대한 신뢰성을 높인다. 시뮬레이션에서는 작업발생기에서 생성된 작업들중 처음과 마지막 5%를 제외한 50000개의 작업을 사용하며, 큐에 도착하는 작업의 수는 작업도착율에 따라 달라진다. 포아송 분포에서는 작업의 평균도착율만이 주어지기 때문에 각 작업의 도착시간을 알 수 없으므로큐에서 대기한 작업의 대기시간을 측정하기 어렵다. 그러나 본 논문에서는 각 작업이 큐에 도착할 때와 할당될 때의 시스템 시간을 읽어서 작업의 대기시간을 계산한다. 그리고 작업도착율 즉, 시스템 부하(system load)에 따른 각 작업할당 방법들의 평균작업수행비용(mean job execution time)을 구한다. 이 평균작업수행비용은 각 작업할당 방법의 성능을 평가하는 기준으로 다음과 같이 정의한다.

$$\text{평균작업수행비용} = \frac{\sum \text{작업수행비용}}{\text{총 작업 수}} \qquad (5)$$

먼저, 평균작업수행시간()이 5 단위시간이고, 표준편차(σ)가 1인 정규분포 $N(\mu, \sigma)$를 따르는 수행시간을 가진 작업들을 기존의 버디 할당방법, 적응적 작업할당 방법(RSR-1, RSR-3), 그리고 Estimate-fold 작업할당 방법에 할당하여, 시스템 부하에 따른 평균작업수행비용을 구하면 (그림 6)과 같다.

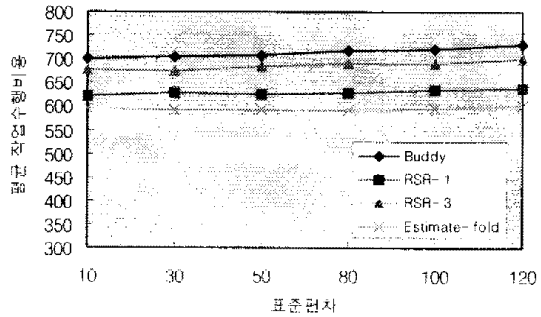


[작업크기 : N(16, 4), 작업수행시간 : N(5, 1)]

(그림 6) 시스템 부하에 따른 평균작업수행비용

(그림 6)의 결과를 분석해 보면, 시스템 부하가 0.4 이하에서는 작업크기 축소를 고려한 작업할당 방법(RSR-t, Estimate-fold)과 작업크기 축소를 고려하지 않는 버디 할당방법의 평균작업수행비용의 차가 크지 않다. 그러나 시스템 부하가 0.4이상으로 증가하면서 작업크기 축소를 고려한 작업할당 방법(RSR-t, Estimate-fold)의 성능이 우수하게 나타난다. 그러므로 시스템 부하가 큰 시스템의 성능 개선 방법으로는 작업크기를 축소하여 할당하는 방법이 효과적임을 알 수 있다. 그리고 작업크기를 축소하여 할당하는 방법중에서도 Estimate-fold 작업할당 방법이 가장 효과적인 성능 개선 방법임을 알 수 있다. 또한 작업크기 축소를 고려한 RSR-3과 RSR-6할당방법의 결과를 분석해 보면 폴딩 횟수가 시스템 성능에 미치는 영향력의 정도를 알 수 있다. 즉, RSR-6 할당방법의 평균작업수행비용이 RSR-3할당방법 보다 크다. 이 결과는 폴딩을 많이 한다고 해서 반드시 좋은 결과를 얻는다는 것이 아님을 입증해 준다.

기존의 적응적 작업할당 방법에서는 작업을 수행하기에 적합한 서브큐브가 형성되지 않으면 무조건 폴딩하여 할당한다. 이 때 폴딩으로 인해 증가하는 작업의 수행시간이 감소하는 대기시간보다 클 경우 전체작업 수행비용이 증가하여 시스템 성능이 저하되는 문제점을 검증한다. 이러한 문제는 작업 수행시간이 다양할수록 발생할 확률이 높기 때문에 작업의 평균작업수행시간(μ)을 500 단위시간으로 고정하고 표준편차(σ)를 10~120사이의 값으로 변화시키면서 시뮬레이션 한 결과는 (그림 7)과 같다.



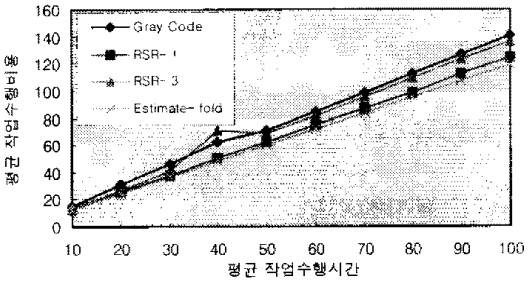
[시스템 부하 : 0.5~0.6, 작업크기 : N(16, 4)]

(그림 7) 표준편차(σ)에 따른 평균작업수행비용

(그림 7)을 분석해 보면, 다양한 크기의 수행시간을 가진 작업이 입력될 때 기존의 적응적 작업할당 방법(RSR-1, RSR-3)은 표준편차의 변화에 따라 평균작업수행비용이 달라진다. 그러나 Estimate-fold 작업할당 방법은 표준편차의 변화에 관계없이 거의 일정한 평균

작업수행비용을 가지므로 기존의 적응적 작업할당 방법(RSR-1, RSR-3)보다 우수한 할당방법임을 입증하고 있다. 이것은 Estimate fold 작업할당 방법이 폴딩하지 않고 적합한 서브큐브가 형성될 때까지 기다리는 경우의 작업수행비용과 폴딩하여 할당하는 과정을 반복하며 각각의 단계별로 구한 작업수행비용을 비교하여 그 중에서 최소 작업수행비용을 갖는 작업할당을 선택하기 때문이다.

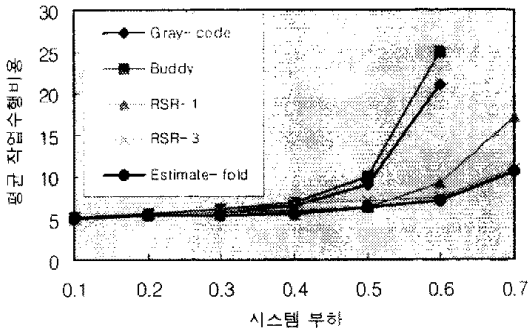
그리고, 평균작업수행시간의 변화에 따른 각 작업할당 방법의 평균작업수행비용을 구한 (그림 8)을 살펴보면 평균작업수행시간 증가에 따라 각 작업할당 방법의 평균작업수행비용도 선형적으로 증가함을 알 수 있다. Estimate-fold 작업할당 방법은 항상 최소의 평균작업수행비용을 가지므로 다른 작업할당 방법에 비해 성능이 우수함을 알 수 있다. 그리고 평균작업수행시간이 40에 가까운 값을 가질 때 RSR-3 알고리즘의 평균작업수행비용이 그레이코드 할당방법보다 큰 값을 가지는 현상은 기존의 적응적 작업할당 방법의 문제점을 보여주는 예이다. 즉 평균작업수행시간이 40에 가까운 작업을 폴딩하여 할당하면 감소하는 대기시간 보다 폴딩으로 인해 증가하는 수행시간이 크기 때문에 전체작업수행비용이 증가하여 일시적으로 시스템의 성능이 저하되는 예를 보여주고 있다.



[시스템 부하 : 0.5~0.6, 작업크기 : (16, 4), 작업수행시간 : (μ, 3)]

(그림 8) 작업의 평균수행시간에 따른 평균작업수행비용

일반적으로 그레이코드 작업할당 방법은 단순하고 서브큐브 인식율이 낮은 버디 작업할당 방법에 비해 우수한 성능을 가진다[6]. 그러나 버디 작업할당 방법에 작업크기 축소를 고려한 작업할당 알고리즘((RSR-1, RSR-3, Estimate-fold)을 적용하면 그레이코드 할당방법 보다 우수한 성능을 얻을 수 있다는 것을 (그림 9)에서 볼 수 있다.



[작업크기 : (16, 4), 작업 수행시간 : (5, 1)]

(그림 9) 작업크기 축소를 적용한 Buddy 할당방법들과 그레이코드 할당방법의 성능 비교

(그림 9)에서 시스템 부하가 증가 할수록 작업크기 축소를 고려한 적응적 작업할당 방법(RSR-1, RSR-3)과 Estimate-fold 작업할당 방법의 성능이 그레이코드 할당방법에 비해 우수함을 보인다. 그러므로 단순한 버디 작업할당 방법에 작업크기 축소를 고려한 RSR-1, RSR-3, Estimate-fold 방법을 적용하면 그레이코드 할당방법 보다 우수한 성능을 얻을 수 있음을 알 수 있다. 그 중에서도 Estimate-fold 작업할당 방법은 모든 시스템 부하 영역에서 최소의 평균작업수행비용을 나타냄으로써 가장 효율적인 작업할당 방법임을 알 수 있다.

5. 결 론

기존의 적응적 작업할당 방법에서는 작업을 수행하기에 적합한 서브큐브가 존재하지 않을 경우 무조건 작업의 크기를 축소하여 할당하기 때문에 작업크기 축소로 인해 증가한 작업수행시간이 대기시간 보다 클 경우 오히려 전체작업수행비용이 증가하는 문제점이 있었다.

본 논문에서는 기존의 적응적 작업할당 방법의 문제점을 개선하기 위해 다음과 같은 세가지 사항을 고려하였다. 첫째, 폴딩으로 인해 증가하는 작업의 수행시간을 고려하였다. 둘째, 작업을 폴딩하지 않고 대기할 경우 그 대기시간 만큼 후속 작업의 할당이 늦어지는 지연시간을 고려하였다. 셋째, 폴딩을 할 것인가에 대한 여부는 폴딩으로 증가하는 수행시간과 감소한 대기시간을 비교하여, 대기시간이 수행시간보다 클 경우에만 폴딩하였다.

이러한 세가지 사항들을 고려한 Estimate-fold 작업

할당 방법은 작업을 수행하기에 적합한 서브큐브가 존재하지 않으면 먼저 작업크기를 축소하지 않고 적합한 서브큐브이 형성될 때까지 기다리는 경우의 작업수행비용을 구한다. 그리고 작업크기를 축소하여 할당하는 과정을 반복하며 각각의 단계별 작업수행비용을 구한다. 이 방법으로 구한 작업수행비용들 중에서 최소 작업수행비용을 갖는 작업할당을 선택함으로써 기존의 적응적 작업할당 방법의 성능을 개선하였다.

시뮬레이션에서는 기존의 적응적 작업할당 방법과 버디 할당방법의 성능을 비교한 시뮬레이션 환경과 동일한 조건하에서 Estimate-fold 작업할당 방법을 시뮬레이션하여 그 결과에 대한 신뢰성을 높였다. 그리고 시뮬레이션을 통하여 기존의 적응적 작업할당 방법에서 폴딩으로 인해 증가하는 수행시간 때문에 전체작업 수행비용이 오히려 증가하는 문제점을 검증하였고, 단순하고 서브큐브 인식율이 낮은 버디 작업할당 방법에 작업크기 축소를 고려한 작업할당 방법(RSR-1, RSR-3, Estimate-fold)을 적용하면 그레이코드 할당방법 보다 우수한 성능을 얻을 수 있음을 알았다. 그리고 작업크기 축소를 고려한 작업할당 방법중에서도 Estimate-fold 작업할당 방법이 가장 효과적인 성능 개선 방법임을 입증하였다.

참 고 문 헌

[1] C. Y. Chang and P. Mohapatra, "An Adaptive Job Allocation Method for Multicomputer Systems," Int. Conf. on Distributed Computing Systems, pp.224-231, May 1996.

[2] V. Lo, J. Kurt, W. Liu and B. Nitzberg, "Noncontiguous Processor Allocation Algorithm for Mesh-Connected Multicomputers" IEEE Trans. on Parallel and Distributed Systems, Vol.8, No 7, pp.712-726, Jul. 1997.

[3] G. Kim and H. Yoon, "On Submesh Allocation for Mesh Multucomputers : A Best-Fit Allocation and a Virtual Submesh Allocation for Faculty Meshes," IEEE Trans. on Parallel and Distributed Systems, Vol.9, No.2, pp.175-185, Feb. 1998.

[4] K. Windish, W. Lo and B. Bose, "Continuous and Noncontiguous Processor Allocation Algorithm for k-ary n cubes," Tech. Report, Oregon State Univ. 1996.

[5] S. Rai, J. L. Trahan and T. Smailus, "Processors Allocation in Hypercube Multiprocessors," IEEE Trans. on Parallel and Distributed Systems, Vol.6, pp.606-616, Jun. 1995.

[6] P. Krueger, T. H. Lai and V. A. Dixit-Radiya, "Job Scheduling is more important than processor Allocation for Hypercube Computers," IEEE Trans. Parallel and Distributed Systems, Vol.5, pp.488-497, May 1994.

[7] J. Kim and C. R. Das, "A Top-Down Processor Allocation Scheme for Hypercube Computers," IEEE Trans. on Parallel and Distributed Systems, Vol.2, pp.20-30, Jan. 1991.

[8] M. S. Chen and K. G. Shin, "Processor Allocation in an N-cube Multiprocessor Using Gray Codes," IEEE Trans. on Computers, Vol.C-36, pp.1396-1407, Dec. 1987.

[9] S. Dutt and J. P. Hayes, "Subcube Allocation in Hypercube Computers," IEEE Trans. on Computers, Vol.40, pp.341-352, Mar. 1991.

[10] H. Wang and Q. Yang, "Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors," Int. Conf. on Parallel processing, pp.25-32, Aug. 1991.

[11] M. Minsky and S. Papert, "On some associative, parallel and analog computations," in Associative Information Technologies, E. J. Jacks, Ed. New York : Elsevier North Holland, 1971.

[12] G. Manimaran, C. S. R. Murthy, "An Efficient Dynamics Scheduling Algorithm For Multiprocessor Real-Time Systems," IEEE Trans. Parallel and Distributed Systems, Vol.9, No.3, pp.312-319, Mar. 1998.

옥 기 상

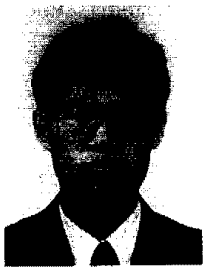
e-mail : ksok@tmrl.kotel.co.kr

1995년 한양대학교 전자계산학과 졸업(학사)

1997년 한양대학교 대학원 전자계산학과(석사)

1997년~현재 한국통신 연구개발본부 통신망연구소 전임연구원

관심분야 : 병렬처리, 작업할당정책, 성능분석, 네트워크 토폴로지



박 준 석

e-mail : jspark@para1.hanyang.ac.kr

1993년 한양대학교 전자계산학과 졸업(학사)

1995년 한양대학교 대학원 전자계산학과(석사)

1998년 한양대학교 대학원 전자계산학과(박사과정 수료)

관심분야 : 컴퓨터구조, 병렬처리시스템, 성능분석

이 원 주

e-mail : wonjoo@doowon.ac.kr

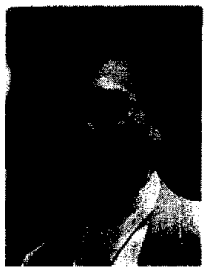
1989년 한양대학교 전자계산학과 졸업(학사)

1991년 한양대학교 대학원 전자계산학과(석사)

1998년 한양대학교 대학원 전자계산학과(박사과정 수료)

1999년~현재 두원공과대학 소프트웨어개발과 전임강사

관심분야 : 병렬처리, 작업할당정책, 성능분석, 광컴퓨터



전 창 호

e-mail : chjeon@para1.hanyang.ac.kr

1977년 한양대학교 전자공학과 졸업(학사)

1986년 Cornell 대학교(박사)

1986년 성균관대학교 전기공학과 조교수

1989년~현재 한양대학교 전자컴퓨터공학부 교수

관심분야 : 컴퓨터 구조, 병렬처리시스템, 성능분석