2차원 매쉬 구조에서 타스크 중복 스케줄링 연구

°전태건 정경훈 김창수
부경대학교 전자계산학과

A Study on Task Duplication Scheduling on 2D-Mesh Topology

°Tae-gun Jeon Kyung-hoon Jung Chang-soo Kim
Department of Computer Science
in Pukyong National University

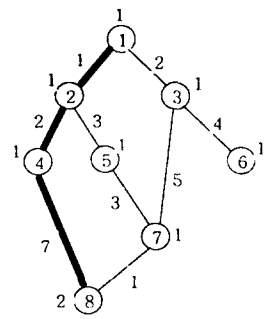
요약

타스크 그래프로 표현되는 병렬 타스크의 수행시간을 단축시키기 위한 타스크 중복 알고리즘의 많은 연구들은 완전연결, 비제한 처리기 상에서 연구되었다. 본 연구에서는 2차원 매쉬구조를 가지는 처리기 상에서의 처리기간 통신 홉 수를 고려한 효율적인 타스크 중복 알고리즘을 제안하고자 한다. 또한 기존의 타스크 중복 알고리즘과 비교를 위해 평균 수행시간과 평균 처리기 사용 수에 대해서 시뮬레이션하였다.

1. 서론

분산·병렬 시스템에서의 처리기간 통신 오버헤드는 타스크 할당 알고리즘에서 가장 중요한 문제중의 하나이며, 이로 인해 병렬 프로그램의 속도향상이 제한되거나 시스템 크기에 비례한 속도향상이 이루어지지 못하는 경우가 많다. 서로 다른 처리기들에 할당된 DAG(Directed Acyclic Graph)내의 두 타스크가 의존관계를 가질 경우 처리기 상호간에 통신 오버헤드가 발생하며 이때 이들 사이의 데이터 교환이 필요하게된다[7]. 타스크 중복은 의존성을 갖는 타스크들 중 일부를 중복하여 병렬 프로그램을 스케줄링함으로써 처리기간 통신 오버헤드를 줄여 전체 수행시간을 향상시키는 방법이다[8]. 이 방법은 프로그램의 전체 수행시간을 향상시킬 수 있는 대기 타스크들의 시작시간을 감소시키며, 특히 통신 오버헤드가 큰 시스템에 유용하다.

일반적으로 최적 타스크 스케줄링 문제는 NP-Complete로 알려져 있기 때문에 제안된 대부분의 연구는 근사 최적인 휴리스틱 방법을 사용하고 있으며, 본 연구도 이 방법을 사용한다. 본 연구와 관련된 휴리스틱 방법[1-4]은 우선순위 리스트 스케줄링과 타스크 중복 등에 대한 연구들이 진행되어왔다. 리스트 스케줄링 방법은 각각의 노드 사이의 통신지연으로 인해 처리기간 통신 지연이 발생하기 때문에 이로 인해 수행시간이 증가할 수 있다. 따라서 통신지연을 발생시키는 노드를 중복(Duplication)함으로써 처리기간 통신 지연을 줄이는 타스크 중복 방법이 제안되었다. 타스크 중복 방법은 부분중복(Scheduling with Partial Duplication: SPD)과 완전 중복(Scheduling with Full Duplication: SFD)으로 분류할 수 있다. Join 노드는 <그림 1>의 n_7, n_8과 같이 진입차수가 2이상인 노드로서, SPD[9]는 가장 긴 수행시간 경로를 가지는 임계 경로(Critical Path: CP)상에 있는 join 노드의 부모에 대해서만 타스크 복사를 시도한다. 따라서 중복 여부 검사에 대한 시간 복잡도는 낮은 반면, 타스크 그래프의 노드 수가 증가하고 깊이가 클수록 처리기간의 통신 오버헤드가 급격히 증가한다는 단점을 가지고 있다. 이에 비해 SFD[9]는 join 노드의 모든 조상노드들에 대해 복사를 시도하기 때문에 SPD에 비해 중복 여부 검사를 위한 시간 복잡도는 높은 반면, 타스크를 최적 처리기에 할당할 수 있다는 장점이 있다. 따라서 본 논문은 SPD의 시간 복잡도와 SFD의 성능 근사한 새로운 타스크 중복 알고리즘을 제안하고자 한다. <그림 1>에서 임계경로는 수행시간 15를 가지는 경로 (n_1, n_2, n_4, n_8)이며, join 노드는 n_7, n_8이다.



<그림 1> CP와 join 노드

2. 기존 연구

2.1 Heavy Node First(HNF) 알고리즘

HNF 알고리즘[1]은 DAG내의 노드들을 레벨(level) 순으로 할당하며, 각 레벨에서 스케줄러는 노드들의 계산비용에 대해 내림차순으로 정렬한다. 동일 레벨에서 두 개 이상의노드들의 실행비용이 같으면 임의의 순으로 노드가 선택되며, 선택된 노드는 가장 빠른 시작시간을 제공하는 처리기에 할당된다.

2.2 DFRN(Duplication First and Reduction Next) 알고리즘

DFRN[9]은 HNF[1]방법을 사용하여 DAG내의 각 노드에 우선순위를 부여한다. Ready 타스크는 ready_queue에 삽입되며, ready_queue내의 노드들은 HNF에 의해 생성된 우선순위 값에 의해 내림차순으로 정렬된다. 만약 노드가 join 노드이면 중복함수를 적용하고 그렇지 않으면 직계부모 노드인 iparent와 같은 처리기에 할당한다.

그러나 이 방법은 join 노드의 모든 조상들에 대해 중복함수를 적용하기 때문에 DAG의 깊이가 증가할수록 시간복잡도가 크게 증가하며, join 노드의 직계 자식 노드들의 수가 많을수록 처리기 사용수가 급격히 증가한다는 단점이 있다. 또한 HNF 방법에 의해 우선순위가 부여되기 때문에 계산 비용은 작으면서 통신 비용이 큰 노드를 고려하지 않았다.

2.3 CPFD(Critical Path First Duplication) 알고리즘

CPFD 알고리즘[6]은 DAG의 노드들을 세가지 종류, 임계경로 노드(Critical Path Node: CPN)와 In-Branch 노드(IBN), Out-Branch 노드(OBN)로 분류한다. CPN은 임계경로상의 노드이다. IBN은 CPN으로의 경로가 존재하는 노드이며, OBN은 CPN도 아니고 IBN도 아닌 노드이다. CPFD는 CPN을 우선적으로 스케줄하려고 한다. 만약 CPN에 대해 스케줄되지 않은 IBN이 존재하면 IBN을 찾아서 먼저 스케줄한다. OBN은 IBN과 CPN이 모두 스케줄된 후에 스케줄된다. CPFD의 기본동기는 CPN을 먼저 스케줄함으로써 병렬시간이 감소될 수 있다는데서 시작된다.

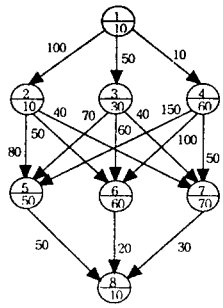
3. 2차원 매쉬 타스크 중복 알고리즘

본 연구에서는 2차원 매쉬 처리기 구조상에서 처리기간 홉 수를 고려한 타스크 중복 알고리즘(MBTD:2D-Mesh, Bounded Task Duplication)을 제안한다.

3.1 가정 및 정의

본 논문에서 사용되는 타스크 그래프의 속성은 DFRN[9]의 가정과 동일하게 설정하고 있다.

알고리즘 기술을 위해서 다음과 같은 정의를 사용한다. 진입(entry) 노드는 부모가 없는 노드이고, 종료(exit) 노드는 자식(child)이 없는 노드이다. <그림 2>에서 노드 1은 진입노드, 노드 8은 종료노드에 해당한다. fork 노드는 진출차수가 2이상인 노드이고 join 노드는 진입차수가 2이상인 노드로써, <그림 2>에서 n_1, n_2, n_3, n_4는 fork 노드이고 n_5, n_6, n_7, n_8은 join 노드이다. iparent(immediate parent: IP)는 노드의 직계부모로서, 노드 4의 iparent는 노드 1이 된다. 관계 $n_i \Rightarrow n_j$는 n_i와 n_j 사이에 강한 선행관계가 있다고 정의한다. 즉, n_i는 n_j의 iparent를 의미한다.



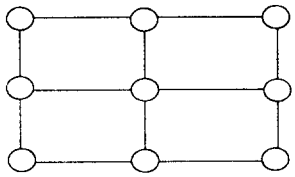
<그림 2> 타스크 그래프

관계 $n_i \rightarrow n_j$는 n_i와 n_j 사이에 약한 선행관계가 있다고 정의하며, 이는 n_i가 n_j의 iparent가 아닌 조상 노드임을 의미한다. <그림 2>에서 노드 1은 노드 4에 대해 $n_1 \Rightarrow n_4$이고, 노드 7에 대해 $n_1 \rightarrow n_7$이다. Earliest Start Time을 의미하는 EST(n, P)는 타스크 n이 처리기 P에서 가장 빨리 시작할 수 있는 시간을 나타내며, Earliest Completion Time을 의미하는 ECT(n, P)는 타스크 n이 처리기 P에서 가장 늦게 종료하는 시간을 나타낸다. $C(n_i, n_j)$는 노드 n_i와 n_j사이의 통신 비용을 의미하며, Massage Arriving Time을 의미하는 MAT(n_i, n_j)는 n_i에서 전송된 메시지가 n_j에 도착한 시간을 나타낸다. 가령, n_i가 처리기 P에 스케줄되면 가정하면 MAT(n_i, n_j)=ECT(n_i,P)+ $C(n_i, n_j)$로 표현될 수 있다. Critical IParent를 의미하는 CIP(n)은 join 노드 n에 대해 가장 큰 MAT를 가지는 iparent이다. Bottom-up Computation을 나타내는 BC(n_i)는 종료노드에서 n_i까지의 통신 비용과 계산 비용의 합을 의미한다. <그림 2>에서 BC(n_7)=10+30+70 = 110이 된다.

처리기 사이의 통신은 2차원 매쉬 구조상에서 메시지 전달(message passing) 방식으로 이루어진다. 이런 경우, 처리기 사이의 홉 수가 2이상 될 수도 있으므로 타스크를 스케줄할 때 "메시지 비용×홉수"만큼의 통신 비용이 생긴다. 따라서 타스크들을 스케줄할 때 처리기 사이의 홉 수를 결정할 필요가 있다.

3.2 처리기간 홉 수 결정 알고리즘

2차원 매쉬는 선형 어레이를 2차원으로 확장한 것으로서 <그림 3>과 같은 구조로 되어 있다. 일반적으로 $m \times n$ 매쉬 구조에서 노드들은 m개의 행과 n개의 열로 구성된다. $m \times n$에서 $m = n$이면 정방매쉬, $m \neq n$이면 사각매쉬라고 한다.



<그림 3> 3×3 매쉬 구조

$m \times n$ 매쉬 구조에서 두 처리기 A와 B사이의 홉 수는 (공식 1)에 의해서 구해질 수 있다.

A와 B사이의 홉 수(H_{AB}) - --(공식 1)

$$= |(x_2 - x_1)| + |(y_2 - y_1)|$$
$$= |int(t_2 \div n) - int(t_1 \div n)| + |(t_2 \% n) - (t_1 \% n)|$$

이때, <그림 3>의 좌측 상단을 (0,0)이라고 하며, 노드 A와 B의 값을 각각 t_1, t_2, 이들의 좌표값을 각각 (x_1, y_1), (x_2, y_2)라 한다.

3.3 MBTD 알고리즘

MBTD 알고리즘의 수행방법은 먼저 DAG내의 가장 긴 임계경로(CP)에 대해 클러스터 하나를 생성한 다음, 나머지 노드들의 BC(Bottom Computation) 값을 계산한다. 그리고 클러스터와 다른 ready 타스크들을 ready_queue에 삽입한 후, BC값에 대해 내림차순으로 정렬한다. 그런 다음, ready_queue에 타스크가 존재하는한 다음을 반복한다: i)FIFO 방식으로 타스크를 선정한다. ii)자신이 join 노드이면 부모 노드의 부모까지만 중복함수를 적용한다. iii)자신이 join 노드가 아닐 경우, CPN이면 P_c에 스케줄하고, 그렇지 않으면 iparent와 같은 처리기 혹은 P_s에 스케줄한다. <그림 2>의 타스크 그래프에 대해 MBTD와 처리기간 홉 수를 고려한 DFRN, CPFD를 적용한 결과가 <그림 4>에 있다. 튜플 P_i:[S,N,F]에서 P_i는 처리기 번호, N은 타스크 그래프내의 노드 번호, S는 노드 N의 시작시간, F는 노드 N의 종료시간을 의미한다.

DFRN
- P_0 : [0,1,10][10,4,70][100,7,170]
- P_1 : [0,1,10][10,3,40]
- P_2 : [0,1,10][10,2,20]
- P_3 : [0,1,10][10,4,70][70,3,100][100,2,110][110,6,170]
- P_4 : [0,1,10][10,4,70][70,3,100][100,2,110][110,5,160][160,7,230][230,8,240]

CPFD
- P_0 : [0,1,10][10,4,70][70,2,80][80,7,150]
- P_1 : [0,1,10][10,3,40][100,5,150][180,8,190]
- P_2 : [0,1,10][10,2,20][100,6,160]

MBTD
- P_0 : [0,1,10][10,4,70][80,7,150][150,5,200][200,8,210]
- P_1 : [0,1,10][10,3,40]
- P_2 : [0,1,10][10,2,20]
- P_3 : [0,1,10][10,4,70][70,3,100][100,6,160]

<그림 4> 타스크 할당 결과

<그림 4>와 같이 MBTD의 수행시간은 210이고 사용된 처리기 수는 4개이다. 기존의 타스크 중복 알고리즘(DFRN, CPFD)과 비교해 볼 때 DFRN보다는 전체 수행시간과 사용된 처리기 수가 작은 반면, CPFD보다는 사용된 처리기 수는 크거나 같으며 전체 수행시간은 작거나 같다. CPFD의 경우 부모노드가 스케줄된 모든 처리기들과 사용되지 않은 하나의 처리기에 대해 중복 알고리즘을 적용하기 때문에 fork 노드의 진출차수와 join 노드의 진입차수가 큰 노드들이 많을수록 타스크 중복을 위한 시간 복잡도가 상당히 크게 증가한다. 따라서 본 논문에서 제안한 MBTD 중복 알고리즘은 부분중복 방법을 사용하기 때문에 DFRN과 CPFD에 비해 타스크 중복 여부검사를 위한 시간 복잡도가 낮았다.

MBTD에 대한 알고리즘은 <그림 5>에 기술되어 있다.

```
단계 1
1 Clustering the CP in DAG    // Initialize DAG
2 Insert the ready tasks into the ready_queue.

단계 2
3 while not empty ready_queue do
4   if n_i is CPN then // Case n_i is the node in CP.
5     if n_i is not JN then
6     Schedule n_i onto P_c   // P_c: the processor scheduled CPNs
7     else
8       try_duplication (P_c, n_i)
9       Schedule n_i onto P_c
10    end if
11  else        // Case n_i isn't the node in CP.
12    if n_i is not JN then //Case n_i isn't join node.
13      identify the IP
14      if the IP is LN then
15        Schedule n_i to the PE having the IP
16      else if exist the unused P_a
17        Copy the scheduled up to IP onto P_a
18        Schedule n_i to the P_a
19      else
20        identify P_s   // P_s : the processor which
                         //has the fastest ready time.
21      if IP was scheduled onto the P_s then
22        Schedule n_i onto the P_s
23      else
24        try_duplication (P_s, n_i)
25        Schedule n_i onto the P_s
26      end if
27    else                // If n_i is join node
28      identify CIP and P_CIP
29      if CIP is LN then // If CIP is the Last Node
30        try_duplication (P_CIP, n_i)
31        Schedule n_i to the P_CIP
32      else
33
34        identify P_s
35        try_duplication (P_s, n_i)
36        Schedule n_i to the P_s
37    end if
38 Insert the ready task into the ready queue.
39 end while

40 try_duplication (P_a, n_i)
41 for each n_p, n_q (MAT(n_p, n_i) ≥ MAT(n_q, n_i),
                      n_p => n_i, n_q => n_i, p≠q,
42                    n_p and are not on P_a yet)
43  if there is any n_x,
      ( real_MAT(n_x, n_p) ≥ real_MAT(n_y, n_p), n_x => n_p,
44    n_y => n_p, x≠y, n_x and n_y are not on P_a yet)
45      EST(n_p, P_a) = Max(real_MAT(n_x, n_p), ST(P_a))
                    // ST(P_a): P_a의 start time
46      Schedule n_p onto P_a
```

```
47  else
48      Schedule n_p onto P_a
49  end if
50      try_deletion (P_a, n_p)
51 end for
52 try_deletion (P_a, n_i)
53  delete any duplicated tasks n_k
                    if ECT(n_k, P_a) > real_MAT(n_k, n_i)
54  real_MAT(n_x, n_p)
55 return ( ECT(P_n_i, n_x) + HC(n_x, n_p) × c(n_x, n_p)
56   // HC(Hop Counter) : The number of Hops
```

<그림 5> MBTD 알고리즘

1에서는 DAG내의 가장 긴 CP만을 클러스터링한다. 2에서는 각 노드의 BC값에 따라 내림차순으로 ready 타스크들을 ready_queue에 삽입한다. 4~10은 CPN(Critical Path Node)을 처리한다. 만약 CPN n_i가 join 노드가 아니면 CPN들이 스케줄된 처리기 P_c상에 스케줄되며, join 노드이면 try_duplication (P_c, n_i)를 수행한 후에 n_i를 P_c에 스케줄한다. 만약 노드 n_i가 CPN이 아니면서 비 join 노드이면 13~26이 수행된다. 즉, n_i의 IP를 식별한 후에 IP가 LN(Last Node)이면 노드 n_i를 IP가 스케줄된 처리기 상에 스케줄한다. 만약 IP가 LN이 아니면 사용되지 않은 처리기 P_a의 존재여부를 확인하여, 존재하면 17, 18과 같이 P_a상에 n_i의 IP까지를 복사한 다음 n_i를 스케줄하고, 존재하지 않으면 20~25가 수행된다. 20~25의 경우, 먼저 시작시간(start time)이 가장 빠른 처리기 P_s를 식별한다(라인 25). 만약 노드 n_i의 IP가 P_s에 스케줄되었다면 n_i를 이 처리기에 스케줄하며, 그렇지 않으면 try_duplication (P_s, n_i)을 수행한 다음 n_i를 스케줄한다. 27~36은 비 CPN이면서 비 join 노드를 처리하는 부분이다. 28은 노드 n_i의 CIP와 P_{CIP}를 식별한다. 여기서 P_{CIP}는 CIP가 스케줄되어 있는 처리기를 의미한다. 만약 CIP가 LN이면 30, 31과 같이 try_duplication (P_{CIP}, n_i)을 수행한 다음, n_i를 P_{CIP}에 스케줄한다. CIP가 LN이 아닐 경우, P_s를 식별한 후에 try_duplication (P_s, n_i)을 수행한 다음, n_i를 P_s에 스케줄한다. 함수 try_duplication (P_a, n_i)의 알고리즘은 <그림 5>의 40~51에 기술되어 있으며, 처리기 P_a에 할당되지 않은 n_i의 grand parent 노드들을 복사한다. 그리고 52에 있는 *try_deletion*(P_a, n_i)을 사용하여 중복된 노드의 삭제여부를 결정한다. 53에서처럼 중복된 노드 n_k의 완료시간 $ECT(n_k, P_a)$가 n_k에서 노드 n_i로의 처리기간 홉 수가 고려된 메시지 전송시간 $real_MAT(n_k, n_i)$보다 크면 중복된 노드 n_k를 삭제한다.

4. 기존 연구와 비교

기존 연구와의 비교를 위해 타스크 그래프는 정적 타스크 그래프를 사용하였으며, 처리기간 통신은 메시지 전달(message passing) 방식을 사용하였다.

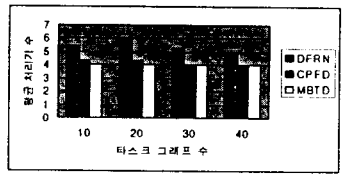
알고리즘 성능은 처리기 사용수와 수행시간에 대해서 평가되었다. 기존의 알고리즘들은 비제한 처리기를 가정하였기 때문에 비교에 이용되는 처리기 사용 수와 수행시간은 평균값을 사용하였으며, 이는 다음 식에 의해 구해질 수 있다.

평균 처리기 수(EPN) =

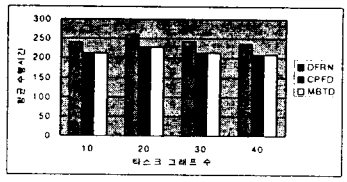
$$\frac{\text{전체 타스크 그래프에 필요한 총 처리기 수}(TPN)}{\text{타스크 그래프 수}(GN)}$$

평균 수행시간(EET) =

$$\frac{\text{전체 타스크 그래프의 총 수행시간}(TET)}{\text{타스크 그래프 수}(GN)}$$



<그림 6> 평균 처리기 사용수 비교



<그림 7> 평균 수행시간 비교

<그림 6>에서와 같이 MBTD 알고리즘은 다른 두 알고리즘에 비해 평균 처리기 사용수가 적거나 같음을 알 수 있다. 이는 기존 알고리즘의 경우 타스크 그래프내에 fork 노드의 수가 많아질수록 사용되는 처리기 수도 증가하는 반면, MBTD 알고리즘에서는 처리기 수를 고정시킨 상태에서 타스크 그래프를 스케줄링 하기 때문에 사용된 처리기 수가 적었다.

<그림 7>은 기존의 중복 알고리즘을 본 연구와의 비교를 위해 통신지연을 처리기간 홉 수배만큼 더하여 얻은 결과로서 MBTD는 DFRN보다는 EET가 작고 CPFD보다는 EET가 작거나 같음을 알 수 있다. 그러나 CPFD의 경우 타스크 그래프의 깊이가 깊을수록, 차수가 큰 fork 노드, join 노드 수가 많을수록 타스크 중복을 위한 시간 복잡도가 본 논문에서 제시한 알고리즘보다 상당히 크다는 단점이 있다.

5. 결 론

본 논문은 2차원 매쉬 처리기 구조에서 처리기간 홉 수를 고려함과 동시에 중복 여부 검사에 소비되는 시간복잡도와 전체 수행시간, 사용된 처리기 수를 줄이는 효율적인 타스크 중복 알고리즘을 제안하였다. 기존의 타스크 중복 메카니즘의 경우 대부분 타스크 스케줄링에 사용되는 처리기 수에 제한을 두지 않았으며, join 노드의 모든 조상 노드들을 중복시킴으로서 타스크 그래프의 깊이(depth)가 깊을수록 중복을 위한 시간 복잡도가 크게 증가하는 문제를 가지고 있었다. 본 논문은 중복을 위한 시간 복잡도를 줄이기 위해 각 타스크의 grand parent 노드까지만 중복 여부를 검사하였으며, 처리기간 통신시 이들 사이의 홉 수를 계산하여 통신 비용에 홉 수배만큼 더함으로써 좀더 현실적인 처리기간 통신을 구현하였다.

참고논문

[1] B. Shirazi, A. R. Hurson, "Scheduling and Load Balancing: Guest Editors' Introduction", Journal of Parallel and Distributed Computing, Dec. 1992, pp. 271-275.

[2] B. Shirazi, A. R. Hurson, "A Mini-track on Scheduling and Load Balancing: Track Coordinator's Introduction", Hawaii Int'l Conf. on System Sciences (HICSS-26), Jan. 1993, pp. 484-486.

[3] M. Y. Wu, A dedicated track on "Program Partitioning and Scheduling in Parallel and Distributed Systems", in the Hawaii Int'l Conference on Systems Sciences, Jan. 1994.

[4] T. Yang and A. Gerasoulis, A dedicated track on "Partitioning and Scheduling for Parallel and Distributed Computation", in the Hawaii Int'l Conference on Systems Sciences, Jan. 1995.

[5] H. El-Rewini, T. G. Lewis, "Task Scheduling in Parallel and Distributed Systems", 1994, Prentice-Hall.

[6] I. Ahmad, Yu-Kwong Kwok, "A new approach to scheduling parallel programs using task duplication", Int'l Conf. on Parallel Processing, 1994, pp. II-47 - II-51.

[7] V. Sarkar, "Partitioning and Scheduling Parallel Programs for Multiprocessor Scheduling", 1989, MIT Press, Cambridge, MA.

[8] T. G. Lewis, H. El-Rewini, "Introduction to Parallel Computing", Prentice-Hall, 1992, New-York.

[9] Gyung-Leen Park, B. Shirazi, "DFRN: A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems", Proceedings 11th International Parallel Processing Symposium, 1997, pp. 157-166.