공유 메모리 다중 프로세서 시스템을 위한 가변 스케줄링

강 오 한†

요 약

본 논문에서는 공유 메모리 다중 프로세서 시스템에서 태스크 중복을 기반으로 하는 휴리스틱 스케줄링 알고리즘을 제안한다. 제안된 알고리즘에서는 공유 메모리에서 통신할 때 발생되는 충돌을 방지하기 위하여 네트웍 통신 자원을 우선 할당하고, 스케줄링 길이를 단축하고 병렬처리 시간을 줄이기 위한 중복 태스크를 선택할 때 휴리스틱을 사용한다. 제안된 알고리즘은 태스크 그래프를 입력으로 받아 다중 프로세서로 스케줄링하며, 시스템에서 사용 가능한 프로세서의 수에 맞도록 태스크를 스케줄링 할 수 있다. 시뮬레이션에서는 제안된 알고리즘을 실제 응용프로그램의 태스크 그래프에 적용하였으며, 프로세서 수의 변화에 따른 스케줄링 길이를 비교하여 제안된 알고리즘의 성능이 우수함을 보여주었다.

S3M2 : Scalable Scheduling for Shared Memory Multiprocessors

Oh-Han Kang†

ABSTRACT

In this paper, a task duplication based heuristic scheduling algorithm is proposed to solve the problem of task scheduling on Shared Memory Multiprocessors (SMM). The proposed algorithm pre-allocates network resources so as to avoid potential communication conflict, and the algorithm uses heuristics to select duplication tasks so as to reduce schedule length and parallel processing time. The algorithm schedules the task of a task graph on to the processors of a multiprocessors, and generates scheduling according to the available number of processors in a system. The proposed algorithm has been applied to some practical task graphs in the simulation, and the results show that the proposed algorithm achieves considerable performance improvement in respect of schedule length.

1. 서 론

병렬 환경에서 시스템의 성능을 향상시키기 위하여 효과적인 스케줄링 알고리즘의 개발이 중요한 연구 분야가 되어왔다. 태스크 스케줄링 알고리즘은 입력된 태스크 중에서 다음에 처리할 태스크를 선택하는 방법이며, 태스크를 프로세서로 할당한다. 병렬 프로그램의 태스크를 효과적으로 스케줄링 함으로서 시스템 자원의 활용도를 높이고 프로그램의 병렬처리 시간을 단축할 수 있다[1, 2]. 다중 프로세서(multiprocessor) 시스템에서 태스크 스케줄링은 NP-complete 문제여서[3] 현재까지 휴리스틱을 기반으로 다양한 알고리즘들이 발표되었다[4-7]. 이들과는 다른 접근 방법으로 다중 프로세서 시스템에서 태스크 중복(duplication)을 기반으로 다양한 스케줄링 알고리즘이 최근에 제안되었다[8-12]. 태스크 중복을 기반으로 하는 스케줄링 알고리즘은 태스크를 다수의 클러스터(cluster)로 나누고 각 클러스터를 프로세서에 할당한다. 또한 각 클러스터에 태스크를 중복하여 할당함으로써 통신을 위한 비용을

※ 본 연구는 안동대학교 기성회 연구비 지원으로 수행되었음
† 정 회 원 안동대학교 시범대학 컴퓨터교육과 교수
논문접수 2000년 1월 11일, 심사완료 : 2000년 9월 19일

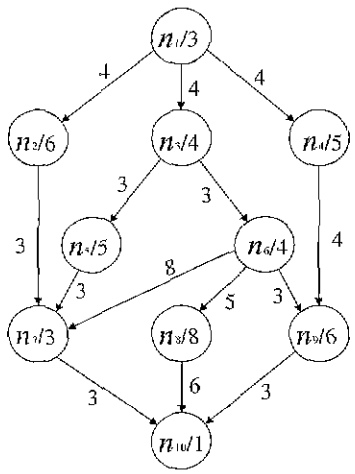
절감할 수 있다. 현재까지 제안된 대부분의 스케줄링 알고리즘들은 완전연결(fully-connected) 네트웍을 가정하였으며 각 프로세서는 독립된 메모리를 가지고 있는 구조를 기반으로 한다.

하드웨어 기술 발전과 통신 기술의 발전에 따른 네트웍의 보급이 확산되면서 다중 프로세서 시스템에 관한 관심이 높아지고 있다. 현재까지 다중 프로세서 시스템 환경에서 병렬 연산을 위한 다양한 네트웍 유형(topology)과 프로토콜이 개발되고 사용되지만 SMM에서는 버스(bus)를 기반으로 하는 네트웍 유형이 광범위하게 사용되고 있다. 비공유 메모리 다중 프로세서 시스템과 비교할 때 SMM의 중요한 한계중의 하나가 메모리 접근을 위한 비용(cost)이 높다는 것이다. SMM에서 병렬 연산을 위한 효과적인 스케줄링 알고리즘을 사용함으로서 비용을 절감할 수 있다. 본 논문에서는 태스크 중복을 기반으로 버스 기반의 SMM에서 적용할 수 있는 휴리스틱 스케줄링 알고리즘을 제안한다. 제안된 태스크 스케줄링 알고리즘은 중복할 태스크를 선택할 때 휴리스틱을 사용하며, 시스템에서 사용 가능한 프로세서의 수에 맞도록 태스크를 스케줄링을 할 수 있다.

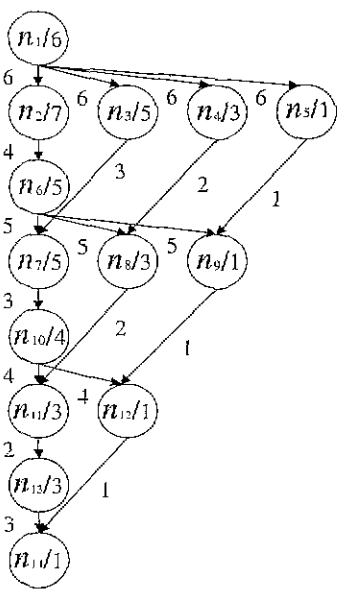
본 논문의 나머지 부분의 구성은 다음과 같다. 2장에서는 태스크 그래프 모델과 용어를 정의한다. 3장에서는 본 논문에서 제안하는 알고리즘을 설명하고, 예제 태스크 그래프를 사용하여 알고리즘의 동작을 설명한다. 4장에서는 제안한 알고리즘의 성능을 평가하기 위한 시뮬레이션 결과를 보여준다. 5장에서는 본 논문에 대한 결론을 나타낸다.

2. 태스크 그래프 및 용어 정의

다중 프로세서 시스템에서 태스크 스케줄링의 주된 목적은 태스크를 서로 다른 프로세서에 할당함으로써 응용 프로그램의 병렬 실행 시간을 단축할 수 있도록 스케줄링 길이를 줄이는 것이다. 이러한 응용 프로그램은 태스크 스케줄링 알고리즘의 입력으로 사용되는 태스크 그래프로 나타낼 수 있다. 태스크 그래프는 V가 태스크 노드이고, E가 통신 링크일 때 튜플(tuple) (V, E, t, c)로 정의할 수 있다. 여기서 집합 t는 연산비용(computation cost)으로 구성되며, 각각의 태스크 $i \in V$는 $t(i)$로 표시하는 연산비용을 갖는다. c는 통신비용(communication cost)의 집합으로 구성되며, 태스크 i에서 태스크 j로 연결되는 링크 $e_{ij} \in E$는 통신비용 c_{ij}를 갖는다. 태스크 그래프에서 링크는 두 태스크 사이의 선행 관계(precedent relation)를 나타낸다. (그림 1)은 태스크 그래프 (Directed Acyclic Graph)의 예를 나타낸 것으로, 태스크 i에 대한 노드 번호는 n_i로 나타낸다.



(a) 일반 태스크 그래프



(b) Cholesky DAG

(그림 1) 예제 태스크 그래프

다음은 제안한 스케줄링 알고리즘을 위한 용어들을 정의하고 관련 수식을 설명한다. 본 논문에서는 태스크 i가 실행을 시작할 수 있는 가장 빠른 시간과 실행을 완료할 수 있는 가장 빠른 시간을 각각 $EST(i)$(Earliest Start Time)과 $ECT(i)$(Earliest Completion Time)로 나타낸다. $ECT(i)$는 $EST(i)$에 태스크 i의 연산비용을 합한 것이다. 태스크 그래프에서 각 노드에 대한 EST의 계산은 시작 노드부터 종료 노드까지 하향식(top-down) 방식으로 진행된다 태스크 i의 모든 부모 노드 j에 대하여 $\{\{ECT(j)+c_{ji}\}\}$ 값이 최대인 노드를 $CPT(i)$(Critical Parent)라고 한다. 테스크 i가 지연되어 완료될 수 있는 가장 늦은 시각과 실행을 시작할 수 있는 가장 늦은 시각을 각각 $LCT(i)$ (Latest Completion Time)와 $LST(i)$(Latest Start Time)로 나타낸다. $LST(i)$는 $ECT(i)$에서 테스크 i의 연산비용을 뺀 것이다. 각 노드에 대한 LST의 계산은 종료 노드에서 시작 노드 방향으로 상향식(bottom-up) 방식으로 진행한다. 태스크 그래프에서 노드 i의 레벨(level)은 노드 i에서 종료 노드까지의 연산비용을 합한 값 중에서 가장 큰 값을 나타낸다. 위의 내용들은 다음과 같은 수식으로 표현할 수 있다.

$$pred(i)=\{j|e_{ji}\in E\} \tag{1}$$

$$suce(i)=\{j|e_{ij}\in E\} \tag{2}$$

$$EST(i)=0 \quad \text{if } pred(i)=\emptyset \tag{3}$$

$$EST(i)=\min\left(\max_{j\in pred(i)\,k\in pred(i),\,k\neq j}(ECT(j),\ ECT(k)+c_{ki})\right) \quad \text{if } pred(i)\neq\emptyset \tag{4}$$

$$ECT(i)=EST(i)+t(i) \tag{5}$$

$$CPT(i)=\{j|(ECT(j)+c_{ji}\geq(ECT(k)+c_{ki})\ \forall j\in pred(i),\ k\in pred(i),k\neq j \tag{6}$$

$$LCT(i)=ECT(i) \quad \text{if } suce(i)=\emptyset \tag{7}$$

$$LCT(i)=\min\left(\min_{j=suce(i),\ i\neq CPT(j)}(LST(j)-c_{ij}),\ \min_{j\in suce(i),\ i=CPT(j)}LST(j)\right) \tag{8}$$

$$LST(i)=LCT(i)-t(i) \tag{9}$$

$$level(i)=t(i) \quad \text{if } suce(i)=\emptyset \tag{10}$$

$$level(i)=\max_{k\in suce(i)}(level(k))+t(i) \quad \text{if } suce(i)\neq\emptyset \tag{11}$$

정의 1 스케줄링이 완료되었을 때 태스크 i의 확정된 시작 시간과 종료 시간을 각각 $RST(i)$(Real Start Time)과 $RCT(i)$ (Real Completion Time)로 정의한다. $RCT(i)$는 $RST(i)$에 태스크 i의 연산비용을 합한 것이다.

정의 2 태스크 i의 $CCPT(i)$(Critical Parent of CPT)는 $CPT(i)$의 CPT로 정의한다. (그림 1a)에서 $CPT(7)$은 n_6 노드이며 $CCPT(7)$은 n_3 노드이다.

정의 3 태스크 그래프에서 모든 태스크의 통신비용의 합한 값을 연산비용의 합한 값으로 나눈 것을 CCR(Communication to Computation Ratio)로 정의한다. (그림 1a)와 (그림 1b) 태스크 그래프에 대한 CCR은 각각 1.24(56/45)와 1.43(69/48)가 된다.

각 노드에 대한 RST와 RCT는 스케줄링 길이를 계산하는데 사용되며 스케줄링이 완료된 후 각 태스크의 실제 시작 시간과 종료 시간을 나타낸다. CPT는 결합(join) 노드를 스케줄링할 때 결합 노드와 같은 프로세서에 할당할 부모 노드를 선택하기 위하여 사용하는 것으로, 결합 노드와 CPT 노드를 같은 프로세서에 할당함으로써 병렬 시간을 줄이기 위한 것이다.

$CCPT$는 현재 스케줄링하고 있는 노드와 CPT 노드의 중복 여부를 결정하기 위한 휴리스틱에서 사용하는 것이다. 위의 수식과 정의에 따라 (그림 1a) 태스크 그래프의 각 노드에 대한 파라미터 값을 계산하면 <표 1>의 결과를 얻을 수 있다.

〈표 1〉 (그림 1a)의 태스크 그래프에 대한 수식의 값

노 드	EST	ECT	LST	LCT	CPT	$CCPT$	$level$
n_1	0	3	0	3	-	-	20
n_2	3	9	6	12	1	-	10
n_3	3	7	3	7	1	-	17
n_4	3	8	3	8	1	-	12
n_5	7	12	7	12	3	1	9
n_6	7	11	8	12	3	1	13
n_7	15	18	15	18	6	3	4
n_8	11	19	13	21	6	3	9
n_9	12	18	12	18	6	3	7
n_{10}	21	22	21	22	8	6	1

3. S3M2 태스크 스케줄링 알고리즘

현재까지 다중 프로세서 환경에서 태스크 중복을 기반으로 하는 다양한 스케줄링 알고리즘들이 제안되었다[8-12, 14]. 이 알고리즘들은 각 프로세서가 독립된 메모리를 가지고 있으며 두 개 이상의 태스크가 동시에 통신할 수 있는 구조를 가정하였다. S. Ranaweera는 최근에 발표한 논문[14]에서 서로 다른 성능을 갖는 다중 프로세서 환경에서 태스크 중복에 의한 스케줄링 알고리즘을 제안하였다. 논문에서는 (그림 1b)에 있는 Cholesky DAG를 사용하여 태스크 완료시간을 비교하고 알고리즘의 우수성을 보여주었다. 현재까지 제안된 논문들이 태스크를 중복한다는 측면에서 본 논문과 유사한 점이 있지만 본 논문에서는 버스 기반의 통신환경을 가정하고 있다. 버스 기반의 통신환경에서는 네트웍 통신 자원의 충돌로 독립된 두 개 이상의 태스크가 동시에 통신할 수 없다. 따라서 모든 프로세서가 동시에 통신이 가능한 것을 가정하고 개발한 이들 알고리즘들은 버스 기반의 SMM에는 적당하지 않다. 본 논문에서 가정하는 버스 기반의 SMM에서는 네트웍에서의 통신 충돌이 병렬 응용 프로그램의 실행시간에 매우 큰 영향을 주며, 네트웍 통신 자원의 동시 사용을 위한 공유가 불가능하다. 따라서 제안된 S3M2 알고리즘에서는 서로 다른 프로세서에 할당된 태스크 사이의 통신을 위하여 네트웍 통신 자원을 우선 배정하여 스케줄링 함으로써 네트웍 충돌을 방지한다.

3.1 태스크 중복을 위한 휴리스틱

S3M2 알고리즘에서는 스케줄링 길이를 단축하기 위하여 중복할 태스크를 선택할 때 휴리스틱을 사용한다. 태스크 그래프에서 결합 노드에 대한 스케줄링이 병렬 시간에 가장 중요한 영향을 미치므로 결합 노드와 어떤 부모(parent) 노드를 동일한 프로세서에 할당할 것인지를 결정하는 것이 중요하다. S3M2 알고리즘의 중요한 개념은 결합 노드와 스케줄링 길이에 결정적인 영향을 줄 수 있는 *CPT* 노드를 선택적으로 중복하여 동일한 프로세서에 배정하므로 스케줄링 길이와 병렬 시간을 단축하는 것이다. 태스크 그래프에서 하나의 노드가 두 개 이상의 노드에 대한 *CPT* 노드로 나타날 수 있다. 이때 *CPT* 노드를 계속적으로 중복하여 클러스터에 할당하는 것은 스케줄링 길이를 연장할 수 있는 가능성이 있다. 따라서 S3M2 알고리즘에서는 휴리스틱을 사용하여 *CPT* 노드를 선택적으로 중복한다. S3M2에서는 다중프로세서 시스템에서 사용된 알고리즘과 같은 방법으로 클러스터를 생성하지만 노드를 스케줄링 할 때 스케줄링 길이를 단축하기 위하여 다음과 같은 휴리스틱을 사용한다

① *CPT* 노드가 현재 스케줄링하고 있는 노드의 유일한 부모 노드이면 *CPT* 노드를 중복한다.
② 스케줄링하고 있는 결합 노드와 동일한 클러스터에 배정할 *CPT* 노드가 이미 다른 클러스터에 배정된 경우에는 결합 노드의 *CCPT* 노드가 결합 노드와 같은 클러스터에 배정될 수 있는 경우에만 *CPT* 노드를 중복하여 배정한다.
③ 결합 노드에서 모든 부모 노드가 이미 다른 프로세서에 배정된 경우에는 *CPT* 노드를 중복한다.

3.2 알고리즘 구조와 동작 예제

태스크 중복을 기반으로 하는 스케줄링 알고리즘들은 스케줄링 길이를 단축하기 위하여 태스크를 여러 개의 프로세서에 중복하여 배정한다. 이들 알고리즘들은 특히 태스크 그래프에서 결합 노드와 결합 노드의 스케줄링에 가장 영향을 주는 부모 노드를 동일한 프로세서에 할당하며, 노드의 중복이 가능하다 다중 프로세서 시스템에서는 각 프로세서의 직접적인 통신이 가능하므로 많은 태스크를 중복함으로써 스케줄링 길이를 단축할 수 있다. 그러나 버스 기반의 공유 메모리 환경에서는 독립된 태스크의 통신 자원에 대한 동시 사용이 불가능하여 태스크 중복으로 인한 스케줄링 지연이 발생될 수 있다.

S3M2 알고리즘의 4개의 단계로 나누어진다 단계 1에서는 2장의 수식과 정의를 사용하여 클러스터 생성에 필요한 값들을 계산한다 <표 1>은 알고리즘에서 (그림 1a)의 태스크 그래프를 사용하는 경우 단계 1의 처리 결과를 나타낸 것이다. 이 단계는 현재까지 제안된 다중 프로세서 환경의 스케줄링 알고리즘과 유사하며, 정의 2와 정의 3에 있는 *CCPT*와 *CCR*을 추가로 계산한다 단계 2에서는 다중 프로세서 환경의 알고리즘과 동일은 방법으로 단계 1에서 구한 값들을 이용하여 중복된 태스크로 구성된 태스크 클러스터를 생성한다. 이와 함께 S3M2 알고리즘의 단계 2에서는 중복할 태스크를 선택하기 위하여 앞 절에서 설명한 휴리스틱을 사용한다. 단계 3에서는 알고리즘에서 필요로 하는

프로세서 수가 시스템에서 사용 가능한 프로세서의 수보다 많으면 태스크 클러스터를 합병하여 클러스터의 수를 줄인다. (그림 2)는 S3M2 알고리즘에서 단계 3을 나타낸 것이다.

```
/* 입력: 1) 중복된 태스크로 이루어진 태스크 클러스터들
         2) num_of_AP: 시스템에서 사용 가능한
            프로세서의 수
            num_of_RP: 알고리즘에서 필요로 하는 프
            로세서의 수
   출력: 1) 합병된 태스크 클러스터들 */

 while(num_of_AP < num_of_RP) {
   calculate exec(i) for each processor i
   /* sum of execution costs of all tasks allocated to
   processor i */
   sort processors in ascending order of exec(i)
   temp = num_of_RP - num_of_AP
   if(temp > num_of_RP/2) temp=num_of_RP/2
   merge tasks of processors j and (temp*2-j-1) for
   j=0 to temp-1
   decrement num_of_RP
 }
```

(그림 2) S3M2 알고리즘의 단계 3

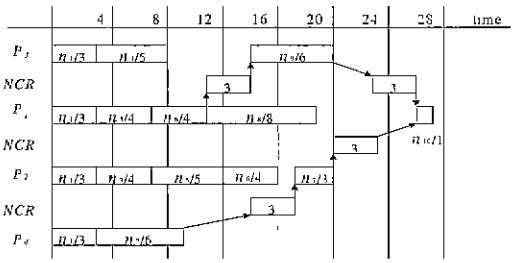
단계 4에서는 세 번째 단계에서 생성한 클러스터를 이용하여 각 태스크의 *RST*과 *RCT*을 확정하고, 응용 프로그램의 스케줄링 길이를 계산한다. 이 단계에서 알고리즘은 버스 기반의 공유 메모리 특성을 고려하여 클러스터에 배정된 태스크를 네트웍 자원의 충돌이 발생되지 않도록 스케줄링한다. 네트웍 통신 자원을 독립된 두 개의 통신 태스크가 동시에 사용할 때 충돌이 발생되므로 스케줄링 알고리즘에서 이를 방지하도록 통신 자원을 할당한다. 이를 위하여 알고리즘에서는 슬롯(slot) 개념을 사용한다. 두 개의 태스크가 통신하는 시간 동안을 하나의 슬롯으로 보며, 프로그램이 시작되기 전에 모든 슬롯은 비어있다고 가정한다. 알고리즘에서는 태스크가 통신하기 전에 통신비용에 해당하는 길이의 사용하지 않는 첫 번째 슬롯을 태스크에 배정한다. 각 클러스터에 저장된 태스크를 레벨이 낮은 것부터 스케줄링하며, 스케줄링 할 태스크의 데이터 선행 관계가 만족되지 않으면 다음 클러스터의 태스크를 스케줄링한다. 스케줄링 할 노드의 모든 부모 노드의 스케줄링이 완료되어 데이터 선행 관계가 만족되는 경우에는 네트웍 통신 자원의 충돌이 발생되지 않도록 태스크를 스케줄링한다. (그림 3)은 S3M2 알고리즘에서 단계 4를 나타낸 것이다.

```
단계 4
/ 입력 1) 중복된 태스크로 이루어진 태스크 클러스터들
       2) bus_slot_list: 네트웍 통신 자원에서 사용 가
          능한 슬롯을 나타냄
            초기값: [0,∞], 형식: [s1, e1], [s2, e2], ,
                    [sn, ∞] */

 let there exit n task clusters on n processors, i.e.
 { P0, P1, , Pn-1}
 while (not all nodes be scheduled) do {
 for (i=0; i<0; i--) {
   select the node ni to be scheduled from Pi
   let there exist p parents of ni, i.e
   { ni1, ni2, ..., nip}
   if (all ni's p parents have been scheduled) {
     for (k=1; k≤p; k++) {
       If ( nik was scheduled onto Pi)
         if (tmp_rst of ni < tmp_rct of nik)
         tmp_rst of ni = tmp_rct of nik
       else {
         let[ si, ei] be the first slot in
         bus_slot_list for nik
         if ((( ei - si + 1) ≥ c ik,i ) && ( si ≥
           RCT of nik)) {
           slot_busy = si + c ik,i
           substitute [ si, ei] to [ si + c ik,i, ei]
         }
         else
           if ((RCT of nik + c ik,i) ≤ ei)) {
             slot_busy = RCT of nik + c ik,i
             substitute [ si, ei] with [ si, RCT of
             nik] and [RST of ni, + c ik,i, ei]
           }
           if (tmp_rst of ni < slot_busy)
             tmp_rst of ni = slot_busy
         }
         if (tmp_rst of ni < Pi_busy) {
           tmp_rst of ni = Pi_busy
           tmp_rst of ni = tmp_rst of ni + c ik,i
           Pi_busy = tmp_rct of ni
         }
     }
   }
   if (RST of ni is not assigned) {
     RST of ni = tmp_rst of ni; RCT of ni =
     tmp_rct of ni }
   else
     if (tmp_rst of ni < RST of ni) {
       RST of ni = tmp_rst of ni, RCT of ni
       = tmp_rct of ni, }
 }
 }
```

(그림 3) S3M2 알고리즘의 단계 4

(그림 4)는 (그림 1a)의 태스크 그래프에 S3M2 알고리즘을 적용하여 스케줄링한 결과를 나타낸 것이다.



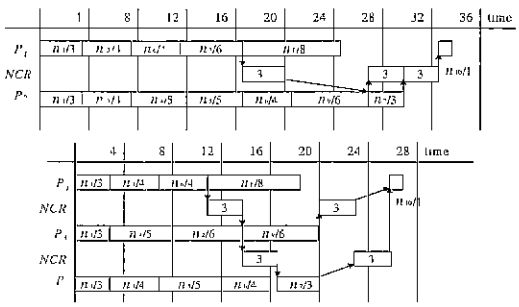
(그림 4) S3M2 알고리즘을 적용한 (그림 1a) 태스크 그래프의 스케줄링

(그림 4)에서 P_i는 프로세서를 나타내며 *NCR*은 네트웍 통신 자원의 사용을 나타낸다. 그림에서 첫 번째 프로세서인 P_1에는 n_{10}, n_8, n_6, n_3, n_1 노드들이 포함된 첫 번째 태스크 클러스터가 배정되어 있으며, 각 노드는 1, 8, 4, 4, 3의 연산비용을 갖는 것을 나타낸다. S3M2 알고리즘을 적용하면 스케줄링 길이는 27이며, 알고리즘에서 필요로 하는 프로세서의 수는 4이다.

S3M2 알고리즘의 단계 1에서는 2장의 정의들을 사용하여 클러스터 생성에 필요한 파라미터 값들을 계산한다. 이 단계는 현재까지 제안된 다중 프로세서 환경의 스케줄링 알고리즘과 유사하다

단계 2에서는 단계 1의 결과를 이용하여 중복된 태스크로 구성된 태스크 클러스터를 생성하며, 각 노드에 대한 중복될 *CPT* 노드를 선택하기 위하여 휴리스틱을 사용한다. 각 노드는 스케줄링 되기 전에 태스크 그래프에서의 레벨을 기준으로 오름차순으로 저장한다. 태스크 클러스터는 태스크 그래프에서 레벨이 가장 낮은 노드에서 시작하여 각 노드의 *CPT* 노드를 동일한 클러스터에 할당하면서 진행한다 각 노드의 중복 여부는 휴리스틱에 의하여 결정되는 것으로, 클러스터 생성을 위해 반드시 필요한 노드와 스케줄링 길이를 줄일 수 있는 노드를 클러스터에 중복한다. 스케줄링하는 현재 노드의 *CPT* 노드가 이미 다른 클러스터에 할당된 경우에는 정의 2의 *CCPT* 개념을 적용하여 *CPT* 노드의 중복을 위한 휴리스틱을 사용한다.

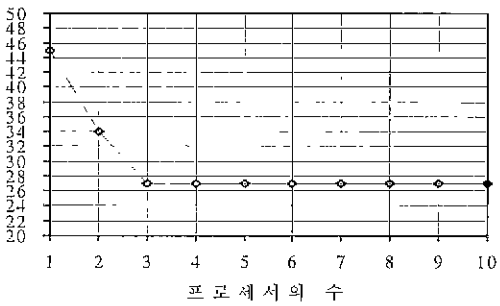
단계 3에서는 사용 가능한 프로세서의 수가 알고리즘에서 요구하는 프로세서의 수보다 적으면 다른 프로세서들로 구성된 태스크 클러스터를 합병한다. 예를 들어 사용 가능한 프로세서의 수가 2개라면 클러스터를 합병하는 과정은 다음과 같다. 각 프로세서 i에 할당된 태스크에 대한 연산비용의 합인 *exec(i)*를 계산한다. 알고리즘에서는 각 프로세서를 *exec(i)*에 따라 오름차순으로 정열하고 그것을 기준으로 logarithmic 형태로 클러스터를 합병한다. (그림 1a)의 태스크 그래프를 사용하면 프로세서 P_1, P_2, P_3, P_4에 대한 *exec(i)* 값은 20, 15, 19, 9이다 따라서 이 예에서는 프로세서 (P_1, P_4)에 있는 태스크들과 (P_2, P_3)에 있는 태스크들이 합병된다. 이 단계에서 프로세서가 반으로 줄고 사용 가능한 2개의 프로세서로 태스크가 합병된다. (그림 5)은 (그림 1a)의 DAG에 대하여 두 개와 세 개의 프로세서가 사용 가능한 경우에 스케줄링을 나타낸 것이다.



(그림 5) 프로세서 수의 변화에 따른 스케줄링

4. 알고리즘의 성능 평가

(그림 1a)의 태스크 그래프에 S3M2 알고리즘을 적용하는 경우 프로세서 수에 따른 스케줄링 길이의 변화는 (그림 6)과 같다. (그림 6)에서 프로세서의 수를 증가하면 스케줄링 길이가 감소하며, 3개 이상의 프로세서를 사용하면 스케줄링 길이가 27로 일정함을 볼 수 있다.



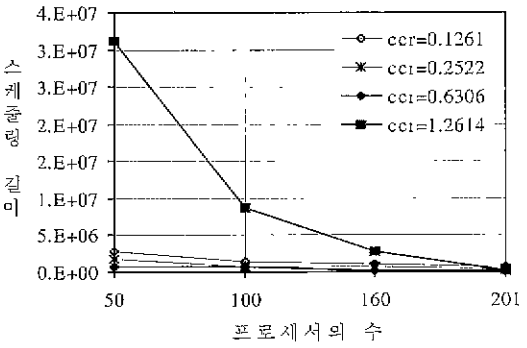
(그림 6) 프로세서 수의 변화에 따른 스케줄링 길이의 변화

본 논문에서 제안한 알고리즘의 성능을 평가하기 위하여 4개의 실제 응용프로그램 태스크를 알고리즘에 적용하였으며, 프로세서 수의 변화에 따른 스케줄링 길이를 비교하였다. 시뮬레이션에서는 ALPES 프로젝트 [13]의 일부분인 Bellman-Ford 알고리즘, Systolic 알고리즘, Master-Slave 알고리즘과 병렬 프로그램의 예로 많이 사용되는 Cholesky decomposition 알고리즘을 사용하였다. (그림 1b)는 14개의 노드로 구성된 Cholesky decomposition DAG의 예를 나타낸 것이다. 각 응용프로그램의 DAG에서 태스크의 수는 2,500개 정도이며, 링크의 수는 4,902에서 20,298개로 구성된다. 부모노드와 자식노드의 수는 1에서 140까지이며 연산 비용은 1에서 20,000 시간 단위까지 변한다. 실험에서 사용한 각 응용 프로그램의 DAG 특성은 <표 2>와 같다.

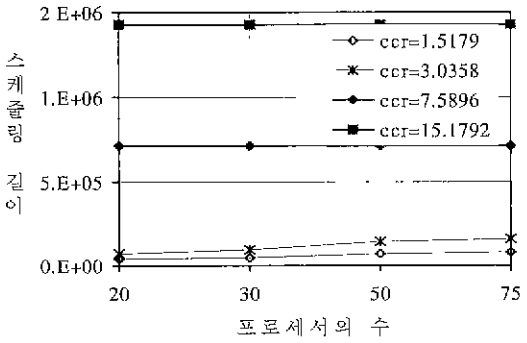
〈표 2〉 응용 프로그램의 성능 특성

특성 \ 알고리즘	Bellman-Ford 알고리즘	Cholesky decomposition	Systolic 알고리즘	Master-slave 알고리즘
노드 수	2,291	2,925	2,502	2,262
링크 수	20,298	5,699	4,902	6,711
CCR	0.1261	1.5179	0.0068	0.0068
알고리즘에서 요구하는 프로세서의 수	201	75	50	54

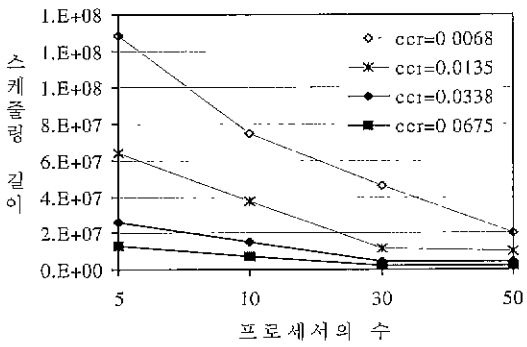
시뮬레이션에서는 통신량의 다양한 변화에 따른 성능을 확인하기 위하여 실제 응용프로그램에서 *CCR*의 값을 1, 2, 5, 10배로 변화시켰다. (그림 7)은 S3M2 알고리즘을 응용프로그램 DAG에 적용할 때 프로세서 수의 변화에 따른 스케줄링 길이의 변화를 나타낸 것이다. (그림 7)에서 프로세서의 수가 증가함에 따라 스케줄링 길이가 감소함을 볼 수 있다. 또한 SMM에서 사용 가능한 프로세서의 수가 알고리즘에서 요구하는 프로세서의 수보다 적을 때에는 사용 가능한 프로세서 수에 맞는 스케줄링 길이를 알고리즘이 생성함을 알 수 있다. (그림 7b)에서 Cholesky decomposition 알고리즘의 스케줄링 길이는 프로세서의 수의 변화에 민감하지 않으며 다른 세 개의 알고리즘은 프로세서의 변화에 따라 스케줄링 길이가 크게 변화됨을 알 수 있다. 그림에서는 *CCR*의 값이 증가함에 따라 Bellman-Ford 알고리즘과 Cholesky decomposition 알고리즘의 스케줄링 길이는 늘어나고 Systolic과 Master-Slave 알고리즘의 스케줄링 길이는 감소함을 보여준다. 이러한 결과는 응용 프로그램의 특성에 의한 것으로 <표 2>에서 그 원인을 확인할 수 있다. <표 2>에서 Bellman-Ford와 Cholesky decomposition 알고리즘의 *CCR*이 Systolic과 Master-slave의 *CCR*보다 상대적으로 매우 큼을 확인할 수 있다. 시뮬레이션에서 사용한 4개의 응용 프로그램에 대한 태스크 그래프는 다양한 특성과 서로 다른 구조적인 특성을 가지므로 4개의 응용프로그램으로부터 동일한 시뮬레이션 결과를 기대하기는 어렵다. 그러나 S3M2 알고리즘은 *CCR* 값의 변화에 관계없이 시스템에서 사용 가능한 프로세서의 수에 맞추어 태스크를 스케줄링 하는 것을 알 수 있다.



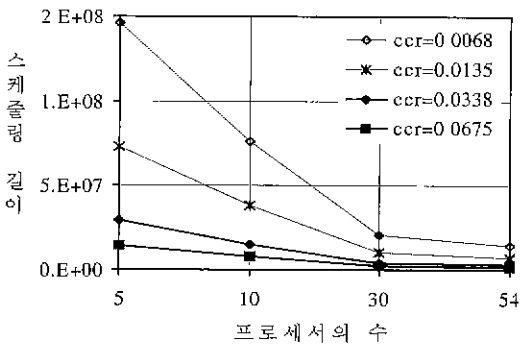
(a) Bllman-Ford 알고리즘



(b) Cholesky decomposition 알고리즘



(c) Systolic 알고리즘



(d) Master-slave 알고리즘

(그림 7) 프로세서 수의 변화에 따른 스케줄링 길이의 변화

5. 결 론

현재까지 다중 프로세서 시스템을 기반으로 하는 다양한 스케줄링 알고리즘이 제안되었으나 네트웍 통신 충돌이 프로그램 실행 시간에 심각한 영향을 주는 공유 메모리 다중 프로세서 시스템 (SMM)에서 그대로 적용할 수가 없다. 본 논문에서는 버스 기반의 SMM에서 적용할 수 있는 태스크 스케줄링 알고리즘을 제안하였다. 제안한 S3M2 알고리즘은 태스크의 중복을 기반으로 하며, 스케줄링 길이를 줄이기 위한 휴리스틱을 사용하여 태스크를 선택적으로 중복한다. 시스템에서 사용 가능한 프로세서 수가 알고리즘에서 요구하는 프로세서의 수보다 적을 경우에 S3M2 알고리즘에서는 사용 가능한 프로세서 수에 맞도록 태스크를 스케줄링한다. 시뮬레이션에서는 실제 응용프로그램의 DAG에 본 논문에서 제안한 스케줄링 알고리즘을 적용하여 프로세서 수의 변화에 따른 스케줄링 길이를 비교하였다.

참 고 문 헌

[1] H. El-Rewini, H H Ali, and T. Lewis, "Task Scheduling in Multiprocessing Systems," *IEEE Computer*, Vol.28, No.12, pp.27-37, 1995

[2] S. Darbha and D. P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, Vol.9, No.1, pp.87-95, 1998.

[3] M. R. Gray and D. S Johnson. "Computers and Interactability : A Guide to Theory of NP-Completeness," W. H. Freeman and Company, 1979.

[4] A. Gereasoulis and T. Yang, "A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors." *Journal of Parallel and Distributed Computing*, Vol.16, pp.276-291, 1992.

[5] C. L. McCreary, A. A. Khan, J. J. Thompson, and M. E. McArdle, "A Comparison of Heuristics for Scheduling DAGs on Multiprocessor," *Proceedings of Eighth International Conference on parallel Processing*, pp.461-451, 1994.

[6] Y-K. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling An Effective Technique for Allocation Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol.7, No.5, pp.506-520, 1996.

[7] T. Yang and A. Gerasoulis. "DSC : Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Transactions on Parallel and Distributed Systems*, Vol.5, No.9, 1994.

[8] I. Ahamd and Y. K. Kwok, "A New Approach to Scheduling Parallel Program using Task Duplication," *Proceedings of International Conference on Parallel Processing*, Vol.II, pp.46-51, 1994.

[9] H. Chen, B. Shirazi, and J. Marquis, "Performance Evaluation of a Novel Scheduling Method - Linear Clustering with Task Duplication," *Proceedings of International Conference on Parallel and Distributed Systems*, pp.270-275, 1993.

[10] S. Darbha and D. P. Agrawal, "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems," *Journal of Parallel and Distributed Computing*, Vol.46, pp.15-26, 1997.

[11] G. L. Park, B Shirazi, and J. Marquis, "DFRN : A New Approach for Duplication Based Scheduling for Distributed Memory Multiprocessor Systems." *Proceedings of Parallel Processing Symposium*, pp. 157-166, 1997

[12] B Shi, H-B. Chen, and J. marquis, "Comparative Study of Task Duplication Static Scheduling versus Clustering and Nonclustering Techniques," *Concur-*

rency : Practice and Experiences, Vol.7, No.5, pp.371-389, 1995.

[13] J. P. Kitajima, and B. Plateau., "Building Synthetic Parallel programs: The Project(ALPES)," *Proceedings of the IFIP WG 10.3 Workshop on Programming Environments ofr Parallel Computing*, Edinburgh, pp.161-170, 1992.

[14] S. Ranaweera and D. P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *Proceedings of 14th Int. Parallel & Distributed Processing Symposium*, pp.445-450, May 2000.

강 오 한

e-mail : ohkang@andong.ac.kr

1982년 경북대학교 전자공학과 졸업(학사)

1984년 한국과학기술원 전산학과 (공학석사)

1992년 한국과학기술원 전산학과 (공학박사)

1984년~1994년 (주)큐닉스컴퓨터 연구원

1994년~현재 안동대학교 컴퓨터교육과 부교수

관심분야 : 병렬처리, 클러스터 시스템, 이동 에이전트 등