버스 기반의 대칭형 다중프로세서 시스템을 위한 태스크 스케줄링 기법

강 오 한† · 김 시 관††

요 약

대칭형 다중프로세서(SMP : Symmetric Multiprocessors) 시스템은 고성능의 병렬 연산을 위한 중요하고 효과적인 기반환경을 제공하고 있다. SMP에서 병렬 태스크와 통신을 위한 스케줄링 기법의 선택은 시스템의 성능에 큰 영향을 미치므로 효과적으로 스케줄링 기법에 대한 연구가 필요하다. 본 논문에서는 버스 기반의 SMP를 위한 태스크 중복을 기반으로 하는 스케줄링 기법을 소개한다. 제안한 스케줄링 기법은 잠재하는 통신 충돌을 방지하기 위하여 네트워크 통신 자원을 사전에 할당한다. 제안한 스케줄링 기법의 성능을 비교하기 위하여 프로세서 수와 통신비용의 변화에 대한 스케줄링 길이를 시뮬레이션을 통하여 조사하였다.

A Task Scheduling Scheme for Bus-Based Symmetric Multiprocessor Systems

Oh-Han Kang† · Si-Gwan Kim††

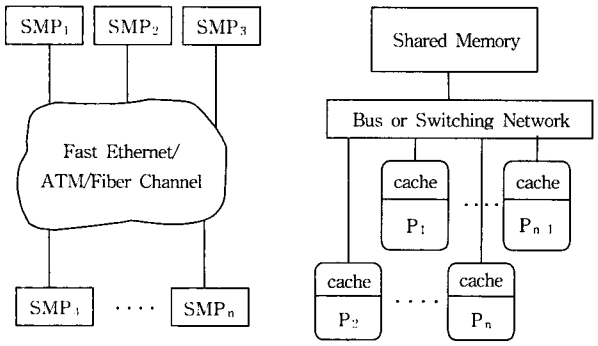
ABSTRACT

Symmetric Multiprocessors (SMP) has emerged as an important and cost-effective platform for high performance parallel computing. Scheduling of parallel tasks and communications of SMP is important because the choice of a scheduling discipline can have a significant impact on the performance of the system. In this paper, we present a task duplication based scheduling scheme for bus-based SMP. The proposed scheme pre-allocates network communication resources so as to avoid potential communication conflicts. The performance of the proposed scheme has been observed by comparing the schedule length under various number of processors and the communication cost.

키워드 : 대칭형 다중프로세서(Symmetric Multiprocessors), 스케줄링(scheduling), 태스크 그래프(DAG), 휴리스틱(Heuristic), 태스크 중복(Task Duplication)

1. 서 론

워크스테이션 클러스터는 가격 측면에서의 장점과 높은 유연성(flexibility)으로 인하여 중요하고 경제적인 병렬연산 환경으로 연구되고 있다[1, 2]. 다양한 고성능 워크스테이션의 출현으로 워크스테이션 클러스터를 사용한 병렬 연산 기반(platform)의 개발이 가능하게 되었다. 향후 워크스테이션은 하나 이상의 프로세서로 구성된 SMP로 개발될 것이며, SMP 시스템은 SMP 클러스터를 구성하여 사용될 것이다. (그림 1)은 SMP 클러스터와 SMP의 구조를 나타낸 것이다. 현재까지 발표된 SMP 클러스터의 예로 DEC Alpha-Server 클러스터[3], SGI Challenge/PowerChallenge[4], Sun Ultra HPC machines, IBM SP system[5] 등이 있다.



(그림 1) SMP 클러스터와 SMP의 구조

최근에는 계층적 메모리 관리, 다중 쓰레드 OS 커널, 컴

※ 본 연구는 한국과학재단 목적기초연구(R05-2000-000-00279-0) 지원으로 수행되었음.
† 종신회원 : 안동대학교 컴퓨터교육과 교수
†† 정 회 원 : 금오공과대학교 컴퓨터공학부 교수
논문접수 : 2002년 2월 22일, 심사완료 : 2002년 10월 10일

파일러 최적화 등 다양한 SMP 관련 기술의 개발이 추진되고 있다. 이와 함께 SMP 환경에서 통신 속도를 개선하기 위한 하드웨어 기술과 통신 프로토콜이 연구되어 왔다. 그러나 SMP 환경의 확산을 위해 해결하여야 할 문제중의 하나가 프로세서 사이의 통신을 위한 비용이 크다는 것이다.

병렬 프로그램의 태스크를 효과적으로 스케줄링 함으로써 시스템 자원의 활용도를 높이고 프로그램의 병렬처리 시간을 단축시킬 수 있다[6-8]. 현재까지 다중 프로세서 시스템을 위한 다양한 태스크 스케줄링 기법들이 제안되었다. 다중 프로세서 시스템에서 태스크 스케줄링은 휴리스틱을 기반으로 하는 기법[9]과 태스크 중복(duplication)을 기반으로 하는 기법이 제안되었다[10, 11]. 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 절감할 수 있다. 현재까지 제안된 대부분의 스케줄링 알고리즘들은 완전연결(fully-connected) 네트워을 가정하였으며 각 프로세서는 독립된 메모리를 가지고 있는 구조를 기반으로 한다.

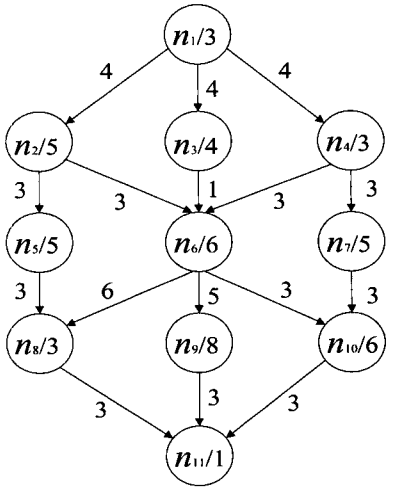
다중 프로세서 시스템과 비교할 때 버스 기반의 SMP 환경에서 중요한 한계중의 하나가 프로세서 사이의 통신을 위한 비용이 크다는 것이다. SMP 환경에서 병렬 연산을 위한 효과적인 스케줄링 알고리즘을 사용함으로서 이러한 문제점을 줄일 수 있다. 버스 기반의 SMP 환경에서 태스크 스케줄링은 완전연결 네트워크에서의 스케줄링과 차이가 있다. 이 환경에서는 네트워크의 통신 자원을 공유할 수 없으므로 서로 독립된 두 개의 통신 태스크가 동시에 수행될 수 없다. 따라서 버스를 기반으로 하는 SMP 환경에서 통신과 동기화 비용은 다중프로세서 시스템보다 상대적으로 매우 크다.

본 논문에서는 버스 기반의 SMP 환경에서 적용할 수 있는 스케줄링 알고리즘을 소개하며 통신비용에 따른 스케줄링 길이의 변화를 조사한다. 알고리즘은 태스크 중복을 기반으로 하며 통신에 따른 스케줄링 길이를 줄이기 위하여 휴리스틱을 사용한다. 시뮬레이션에서는 본 논문에서 소개한 알고리즘을 이용하여 통신비용의 변화에 따른 스케줄링 길이의 변화를 측정하고 분석한다.

본 논문의 2장에서는 태스크 그래프 모델과 용어를 정의한다. 3장에서는 SMP 환경에 적합한 태스크 스케줄링 알고리즘을 설명하고, 예제 태스크 그래프를 사용하여 알고리즘의 동작을 설명한다. 4장에서는 통신비용의 변화에 따른 스케줄링 길이의 변화를 측정한 시뮬레이션 결과를 보여준다. 5장에서는 본 논문에 대한 결론을 나타낸다.

2. 태스크 그래프와 문제 정의

병렬처리를 위한 다중 프로세서 시스템에서 응용 프로그램은 태스크 그래프로 나타낼 수 있다. (그림 2)는 태스크 그래프를 DAG(Directed Acyclic Graph)로 나타낸 예제로서, 태스크 i에 대한 노드 번호는 n_i로 나타낸다. 노드 i에서 '/' 다음의 숫자는 태스크 i의 연산비용을 나타낸다.



(그림 2) 태스크 그래프의 예

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.

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. 태스크 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 태스크 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. 다음은 본 논문에서 제안할 스케줄링 알고리즘을 위한 수식들을 사용한다.

$$pred(i)=\{j \mid e_{ji} \in E\} \quad (1)$$

$$suce(i)=\{j \mid e_{ij} \in E\} \quad (2)$$

$$est(i)=0 \quad \text{if } pred(i)=\Phi \quad (3)$$

$$est(i)=\min\left(\max_{j\in pred(i)\ k\in pred(i),\ k\neq j}(ect(j),\ ect(k)+c_{ki})\right)$$
$$\text{if } pred(i)\neq\Phi \quad (4)$$

$$ect(i)=est(i)+t(i) \quad (5)$$

$$cpt(i)=\{j \mid (ect(j)+c_{ji}\geq(ect(k)+c_{ki})\}$$
$$\forall_j \in pred(i);\ k\in pred(i), k\neq j \quad (6)$$

$$ccpt(i)=cpt(cpt(i)) \quad (7)$$

$$lct(i)=ect(i) \quad \text{if } suce(i)=\Phi \quad (8)$$

$$lct(i)=\min\left(\min_{j\subset suce(i),\ i\neq CPT(j)}(lst(j)-c_{ij}),\ \min_{j\subset suce(i),\ i\neq cpt(j)} lst(j)\right) \quad (9)$$

$$lst(i)=lct(i)-t(i) \quad 10)$$

$$level(i)=t(i) \quad \text{if } suce(i)=\varnothing \quad (11)$$

$$level(i)=\max_{k\in suce(i)}(level(k))+t(i) \quad \text{if } suce(i)\neq\varnothing \quad (12)$$

식 (6)의 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. 수식 (7)에 정의된 태스크 i의 $ccpt(i)$(critical parent of cpt)는 $cpt(i)$의 cpt로 정의한다. 예를 들면 (그림 2)에서 $cpt(8)$은 n_6 노드이며 $ccpt(8)$은 n_2 노드이다.

3. 태스크 스케줄링 알고리즘

Darbha[8]과 같이 현재까지 연구된 다중 프로세서 환경에서 태스크 중복을 기반으로 하는 스케줄링 알고리즘들은 완전 연결 통신망을 기본으로 두 개 이상의 태스크가 동시에 통신할 수 있는 구조를 가정하였다. 따라서 다중 프로세서를 기반으로 하는 이들 알고리즘들은 네트워크 통신 자원의 충돌로 독립된 두 개 이상의 태스크가 동시에 통신할 수 없는 버스 기반의 SMP 환경에는 적당하지 않다. 서로 다른 워크스테이션에 할당된 태스크 사이의 통신을 위하여 네트워크 통신 자원을 우선 배정하여 스케줄링 함으로써 네트워크 충돌을 방지한다. 본 논문에서 제안한 알고리즘에서는 중복할 태스크를 선택할 때 다음과 같은 휴리스틱을 사용함으로써 스케줄링 길이를 단축하여 병렬 시간을 줄인다.

① cpt 노드가 현재 스케줄링하는 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 중복한다.

③ 결합 노드에서 모든 부모 노드가 이미 다른 워크스테이션에 배정된 경우에는 cpt 노드를 중복한다.

위에서 ①번은 어떤 노드의 부모 노드가 하나만 존재하는 경우로 부모 노드를 포함하여야 클러스터의 생성이 가능하다. 따라서 이 부모 노드가 이미 다른 클러스터에 포함되었더라고 cpt 노드인 이 부모 노드를 현재 스케줄링 하는 노드가 포함된 클러스터에 중복한다.

태스크 그래프에서 결합 노드에 대한 스케줄링이 병렬 시간에 가장 큰 영향을 미친다. 위의 휴리스틱에서 ②번과 ③번은 결합 노드와 어떤 부모(parent) 노드를 동일한 프로세서에 할당할 것인지를 결정한다. 알고리즘의 중요한 개념은 결합 노드와 스케줄링 길이에 결정적인 영향을 줄 수 있는 cpt 노드를 선택적으로 중복하여 동일한 프로세서에 배정함으로써 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 본 논문에서는 위의 휴리스틱에서 ②번을 사용하여 cpt 노드를 선택적으로 중복한다. 이 방법은 결합 노드의 cpt 노드 중복을 위하여 cpt 노드의 cpt인 $ccpt$ 노드가 동일한 클러스터에 할당 가능한지를 체크한다. 결합 노드와 $ccpt$ 노드가 동일할 클러스터에 할당될 수 있으면 결합 노드, cpt, $ccpt$ 노드를 하나의 클러스터에 할당한다. 위에서 ③번은 결합 노드에서 모든 부모 노드가 이미 다른 클러스터에 할당된 상태에서 새로운 클러스터 생성을 위하여 부모 노드 중에서 하나를 선택하는 경우이다. 따라서 결합 노드와 cpt 노드를 동일한 클러스터에 할당함으로써 스케줄링 길이를 단축할 수 있다.

3.1 알고리즘 구조

본 논문에서 제안한 스케줄링 알고리즘의 단계 1에서는 2장에서 나타낸 수식들을 이용하여 알고리즘에서 사용할 필요한 파라미터 값들을 구한다. 이 단계는 다중 프로세서 환경의 스케줄링 알고리즘과 유사하다. 단계 2에서는 단계 1에서 구한 값들을 이용하여 중복된 태스크로 구성된 태스크 클러스터를 생성한다. 태스크 클러스터는 태스크 그래프에서 레벨이 가장 낮은 노드에서 시작하여 각 노드의 cpt 노드를 동일한 클러스터에 할당하면서 진행한다. 각 노드의

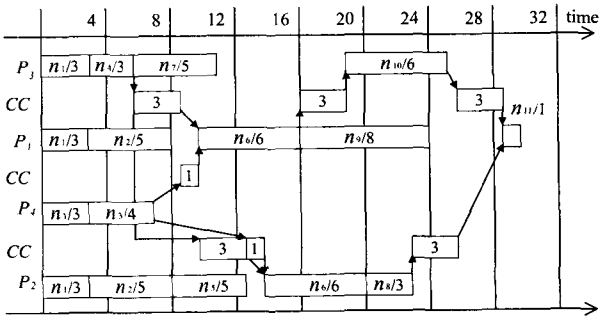
중복 여부는 휴리스틱에 의하여 결정되는 것으로, 클러스터 생성을 위해 반드시 필요한 노드와 스케줄링 길이를 줄일 수 있는 노드를 클러스터에 중복한다. 스케줄링하는 현재 노드의 *cpt* 노드가 이미 다른 클러스터에 할당된 경우에는 식 (7)의 *ccpt* 개념을 적용하여 *cpt* 노드의 중복을 위한 휴리스틱을 사용한다.

단계 3에서는 단계 2의 결과를 이용하여 태스크의 *rst*와 *rct*를 확정하고, 병렬 처리 시간을 예측하기 위한 스케줄링 길이를 계산한다. 이 단계에서 알고리즘은 버스 기반의 SMP 특성을 고려하여 클러스터에 배정된 태스크를 네트워크 자원의 충돌이 발생되지 않도록 스케줄링한다. 네트워크 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 낮은것부터 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 네트워크 통신 자원의 충돌이 발생되지 않도록 스케줄링한다.

3.2 알고리즘의 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 그 노드의 스케줄링에 가장 큰 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 SMP 환경에서는 독립된 태스크의 통신 자원에 대한 동시 사용이 불가능하여 태스크 중복으로 인한 스케줄링 지연이 발생될 수 있다. (그림 3)은 (그림 2)의 태스크 그래프를 사용하여 본 논문에서 제안한 알고리즘을 적용하였을 때 스케줄링한 결과를 나타낸 것이다. 아래에서는 (그림 3)의 스케줄링 과정을 단계별로 설명한다.

(그림 3)에서 P_i는 프로세서를 나타내며 *CC*는 네트워크 통신 자원의 사용을 나타낸다. 그림에서 첫 번째 프로세서인 P_1에는 $n_{11}, n_9, n_6, n_2, n_1$ 노드들이 포함된 첫 번째 태스



(그림 3) 알고리즘의 스케줄링 결과

크 클러스터가 배정되어 있으며, 각 노드는 1, 8, 6, 5, 3의 연산비용을 갖는 것을 나타낸다. 따라서 본 논문에서 제안한 알고리즘을 적용하면 스케줄링 길이는 30이며, 알고리즘에서 필요로 하는 프로세서의 수는 4이다. (그림 3)의 스케줄링 결과를 만들기 위한 알고리즘의 단계별 스케줄링 과정은 다음과 같다.

알고리즘의 단계 1에서는 2장의 수식들을 사용하여 각 노드에 대하여 클러스터 생성에 필요한 값들을 계산한다. $pred(1) = 0$이므로 시작노드의 *est*는 0이다. 따라서 n_1는 시간 3에서 실행을 종료한다. 다른 태스크의 *est*와 *ect*는 수식 (4)와 식 (5)를 사용하여 계산할 수 있다. 각 노드에 대한 *lct*와 *lst*의 계산은 종료 노드에서 시작 노드 방향으로 상향식(bottom-up) 방식으로 진행한다. 식 (1)~식 (12)에 따라 각 노드에 대한 파라미터 값을 계산하면 <표 1>의 결과를 얻을 수 있다.

〈표 1〉 태스크 그래프에 대한 파라미터 값

노 드	*est*	*ect*	*lst*	*lct*	*cpt*	*ccpt*	*level*
1	0	3	0	3	-	-	23
2	3	8	4	9	1	-	20
3	3	7	4	8	1	-	19
4	3	6	3	6	1	-	18
5	8	13	10	15	2	1	9
6	9	15	9	15	2	1	15
7	6	11	7	12	4	1	12
8	15	18	18	21	6	2	4
9	15	23	16	24	6	2	9
10	15	21	15	21	6	2	7
11	24	25	24	25	9	6	1

알고리즘의 단계 2에서는 일반적인 클러스터 알고리즘과 같이 태스크 클러스터를 생성하며, 각 노드의 *cpt* 노드의 중복을 위해서는 앞에서 설명한 휴리스틱을 사용한다. 클러스터 생성을 위하여 모든 노드를 레벨을 기준으로 오름차순으로 정렬하여 큐(queue)에 저장한다. <표 1>에 나타난 레벨을 이용하면 큐에는 $n_{11}, n_8, n_{10}, n_9, n_5, n_7, n_6, n_4, n_3, n_2$

n_1의 순서로 태스크가 저장된다. 큐의 첫 번째 노드인 n_{11}부터 시작하여 각 노드의 *cpt* 노드를 찾아서 하나의 클러스터를 생성한다. 노드 $n_{11}, n_9, n_6, n_2, n_1$들을 포함하는 하나의 클러스터가 생성되며, 이를 프로세서 P_1에 할당한다. 첫 번째 클러스터를 생성하는 과정에서는 중복되는 태스크가 발생되지 않는다.

다음 클러스터의 생성은 큐에서 클러스터에 할당되지 않은 첫 번째 태스크인 n_8부터 시작한다. 결합 노드인 n_8의 *cpt* 노드인 n_6이 첫 번째 클러스터에 포함되어 있으므로 n_8의 다른 부모 노드인 n_5를 사용하여 클러스터를 생성한다. 그 결과 n_8, n_5, n_2, n_1 노드들을 포함하는 두 번째 클러스터가 생성된다. 두 번째 클러스터를 P_2에 할당한다. 결합 노드의 *cpt* 노드인 n_6을 이 클러스터에 포함할 것인지는 휴리스틱을 사용하여 중복 여부를 결정한다. 이 경우는 휴리스틱의 두 번째 조건에 해당하는 것으로, *ccpt*(8)인 n_2가 두 번째 클러스터에 포함되므로 n_6을 이 클러스터에 포함시켜 첫 번째 클러스터와 중복하게 된다. 두 번째 클러스터에 n_2을 중복한 것은 휴리스틱의 첫 번째 조건에 해당하는 것으로 n_5의 부모 노드로 n_2만이 존재하기 때문이다.

세 번째 클러스터의 생성은 n_{10}부터 시작하여 n_{10}, n_7, n_4, n_1 노드로 구성된다. 여기서 n_{10}의 *cpt* 노드인 n_6을 포함하지 않는 것은 *ccpt*(10)인 n_2이 이 클러스터에 포함되지 않았기 때문이다. 네 번째 클러스터는 n_3, n_1로 구성되며, 모든 노드가 클러스터에 할당되어 단계 2가 종료된다. 단계 2에서는 4개의 클러스터를 생성한다. (그림 3)에서와 같이 각 클러스터는 프로세서 P_1, P_2, P_3, P_4에 각각 할당된다. 알고리즘의 두 번째 단계에서 휴리스틱을 사용하여 결합 노드의 *cpt* 노드를 선택적으로 중복함으로서 스케줄링 길이를 단축할 수 있으며 클러스터의 수를 줄일 수 있다.

세 번째 단계에서는 두 번째 단계에서 생성한 클러스터를 이용하여 각 태스크의 *rst*과 *rct*을 확정하고, 응용 프로그램의 스케줄링 길이와 병렬 시간을 계산한다. 각 노드에 대한 *rst*와 *rct*는 스케줄링 길이를 계산하는데 사용되며 스케줄링이 완료된 후 각 태스크의 실제 시작 시간과 종료 시간을 나타낸다. 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.

이 단계의 스케줄링에서 각 태스크가 실행되기 위해서는 태스크 사이의 데이터 선행 조건을 만족하여야 한다. 따라서 각 클러스터에서 입력 노드부터 스케줄링하며 데이터 선행 조건을 만족하지 못하는 경우에는 다음 클러스터의 태스크를 스케줄링 한다. (그림 3)에서 첫 번째 클러스터가 할당된 P_1의 n_1부터 스케줄링하며, 이어서 n_2, n_6, n_9, n_{11} 노드들을 순서대로 스케줄링 하면 된다. 각각의 태스크는 실행 시각에 부모 노드들이 모두 실행되어 데이터 선행 조건을 만족하여야 스케줄링이 가능하고, 동일한 프로세서에 할당되어 통신비용이 소요되지 않는다. n_2의 부모 노드인 n_1의 스케줄링이 완료되었으므로 n_2를 스케줄링 한다. 다음에 스케줄링 할 노드는 n_6이다. n_6을 스케줄링하기 위해서는 n_2, n_3, n_4의 스케줄링이 완료되어야 하나 n_3과 n_4의 스케줄링이 완료되지 않아서 다음 클러스터의 태스크를 스케줄링 한다.

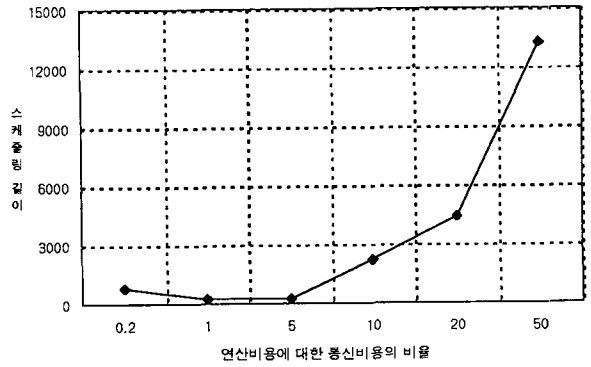
두 번째 클러스터가 할당된 P_2의 n_1, n_2, n_5 노드들을 순서대로 스케줄링하면 된다. n_1, n_2, n_5 노드의 *rst*가 각각 0, 3, 8이 된다. n_5를 P_2에 스케줄링 할 수 있으며, n_6는 부모 노드인, n_3, n_4가 처리되지 않아서 스케줄링이 지연된다. 이어서 P_3에 할당된 n_1, n_4, n_7 노드들을 각각 *rst* 값 0, 3, 6으로 스케줄링 한다. 다음에 스케줄링 할 노드는 스케줄링되지 않은 부모노드로 n_6을 가지고 있는 n_{10}이다. 따라서 n_{10}은 n_6이 완료될 때까지 지연된다.

이어서 마지막 클러스터가 할당된 P_4의 n_1, n_3가 스케줄링되며, P_1에 할당된 태스크의 스케줄링을 다시 시도한다. 3개의 부모노드를 갖는 n_6가 스케줄링 될 순서이다. 부모노드 n_3와 n_4의 스케줄링이 완료되었으므로 n_6가 스케줄링 될 수 있다. 노드 n_6와 n_2는 하나의 프로세서 P_1에 할당되었으므로 두 노드간의 통신비용은 0이 된다. n_6는 n_4, n_3와 통신하기 위하여 네트워크 통신 자원을 시간 6에서 10까지 사용한다. n_6과 n_9가 P_1에 스케줄링되며 *rst* 값은 각각 10과 16이 된다. 다음에 스케줄링 될 노드는 스케줄링 되지 않은 노드 n_8과 n_{10}을 부모노드로 갖는 n_{11}이다. 데이터 선행조건을 만족하기 위해서 n_{11}은 n_8과 n_{10}이 스케줄링 될 때까지 지연된다. 이어서 P_2에 있는 노드 n_6가 스케줄링 될 수 있다. P_2에 있는 n_6는 부모노드인 n_2, n_3, n_4의 스케줄링이 완료되었으므로 바로 스케줄링 될 수 있으며, n_6의 *rst*와 *rct* 값은 각각 14와 20이다. n_3로부터 n_6로의 통신은 태스크 4와 6의 통신이 완료된 후인 시간 13에서 시작된다. n_8이 P_2에 스케줄링 된 후에 P_3의 n_{10}이 스케줄링 될 수 있다. n_6에서 n_{10}로의 통신은 시간 16에서 시작될 수 있으며 시간 16에서 19까지 통신 채널을 사용한다. 따라서 n_{10}의 *rst*와 *rct*는 각각 19와 25가 된다. P_3와 P_4에 할당된 노드들의 스케줄링이 완료되었으므로 P_1의 n_{11}이 다음에 스케줄링 될 수 있다. n_{11}은 n_8, n_{10}과 통신이 필요하다. n_8에서 n_{11}로의 통신은 시간 23에서 시작되며 26까지 통신 채널을 사용하게 된다. n_{10}에서 n_{11}로의 통신은 시간 26에서 시작

될 수 있다. 따라서 n_{11}의 rst와 rct는 각각 29와 30이 된다.

4. 시뮬레이션 결과

본 논문에서는 시뮬레이션을 하기 위하여 SUN 워크스테이션에서 C언어를 사용하여 스케줄링 알고리즘을 구현하였다. 시뮬레이션에서는 프로세서와 ccr의 변화에 따른 스케줄링 길이의 변화를 측정하기 위하여 임의의 태스크 그래프와 실제 응용 프로그램의 태스크 그래프를 입력으로 사용하였다. 태스크 그래프에서 태스크의 통신비용의 합한 값을 연산비용의 합한 값으로 나눈 것을 ccr (communication to computation ratio)로 정의한다. 시뮬레이션에서 사용한 임의의 태스크 그래프는 40개의 노드로 구성되었으며 7개 프로세서를 사용하였다. 시뮬레이션에서는 ccr의 값을 증가시키면서 스케줄링 길이의 변화를 측정하였다. (그림 4)는 ccr의 증가에 따른 스케줄링 길이의 변화를 나타낸 것이다.

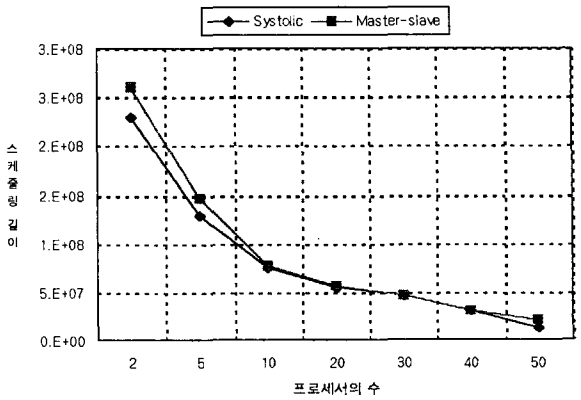


(그림 4) ccr에 따른 스케줄링 길이의 변화

(그림 4)에서 ccr의 값이 증가하면 스케줄링의 길이가 증가함을 확인할 수 있으며, ccr 값의 증가율이 커지면 스케줄링 길이도 급격히 증가함을 알 수 있다. 따라서 SMP 환경에서 통신량의 증가는 병렬 프로그램을 처리하는 시스템의 성능에 큰 영향을 준다고 할 수 있다.

두 번째 시뮬레이션에서는 2개의 실제 응용 프로그램의 태스크 그래프를 사용하여 실험하였으며 프로세서의 수와 ccr의 변화에 따른 스케줄링 길이를 측정하였다. 시뮬레이션에서는 ALPES 프로젝트 [12]에서 사용된 Systolic과 Master-Slave 알고리즘의 DAG를 사용하였다. 태스크 그래프에서 태스크의 수는 각각 2,502와 2,262개이며, 링크의 수는 각각 4,902와 6,711개이다. 이들 태스크 그래프의 초기 ccr 값은 각각 0.068과 0.076이다.

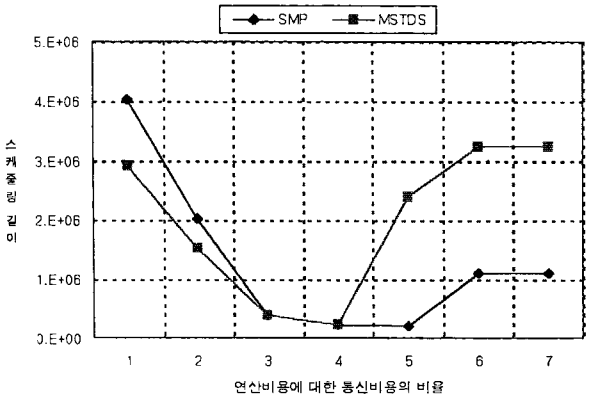
시뮬레이션에서는 먼저 프로세서 수에 따른 스케줄링 길이의 변화를 알아보았다. (그림 5)는 본 논문의 알고리즘을 응용프로그램 DAG에 적용할 때 프로세서 수의 변화에 따른 스케줄링 길이의 변화를 나타낸 것이다. 여기서 ccr의 값은 태스크 그래프의 초기값인 0.068과 0.076을 적용하였다. 두 응용 프로그램에서 프로세서의 수가 증가함에 따라 스케줄링 길이가 감소함을 (그림 5)에서 확인할 수 있다. 시스템에 사용 가능한 프로세서의 수가 많으면 태스크를 분산하여 할당함으로써 스케줄링 길이를 단축할 수 있다.



(그림 5) 프로세서 수에 따른 스케줄링 길이의 변화

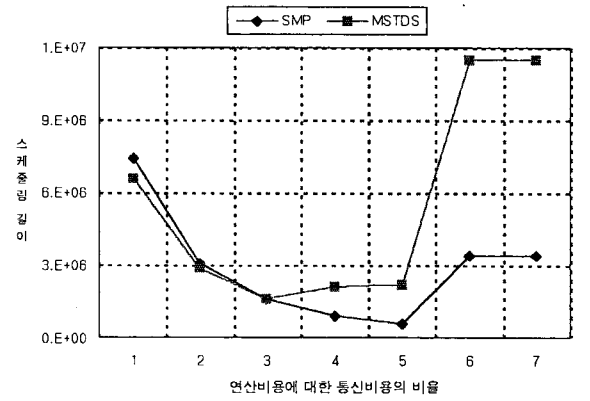
통신량의 다양한 변화에 따른 알고리즘의 성능을 확인하기 위하여 ccr의 값을 변화시키면서 스케줄링 길이를 측정하였다. 본 논문에서 제안한 스케줄링 알고리즘의 성능을 비교하기 위하여 태스크 중복기반의 대표적인 스케줄링 기법인 Darha[8]의 STDS 알고리즘을 사용하였다. STDS 알고리즘이 두 개 이상의 프로세서가 동시에 통신할 수 있는 다중 프로세서 환경을 가정하고 있으므로 이를 버스기반에 적용될 수 있도록 수정하고 이를 MSTDS라 하였다. (그림 6)과 (그림 7)은 시뮬레이션의 결과를 나타낸 것이며 사용된 프로세서의 수는 50개이다. (그림 6)는 Systolic 알고리즘 DAG를 사용하여 ccr의 변화에 따른 스케줄링 길이의 변화를 나타낸 것이다. (그림 6)에서 ccr의 값이 매우 적어지거나 경우에는 스케줄링의 길이가 늘어나는 것을 알 수 있다. 이것은 태스크 중복을 기본으로 하는 스케줄링 알고리즘의 공통적인 특성이다. (그림 6)에서 ccr이 증가함에 따라 스케줄링의 길이가 감소하며 ccr이 일정한 값 이상으로 증가하면 스케줄링 길이도 급격히 증가함을 볼 수 있다. (그림 6)에서 확인할 수 있는 중요한 내용은 ccr이 큰 경우에 본 논문에서 제안한 SMP 알고리즘의 성능이 MSTDS보다 우수하다는 것이다. 이러한 성능의 차이는 SMP 알고리즘에서 결합노드를 스케줄링 할 때 cpt 노드를 선택적으로 중복하는 휴리스틱에 기인한 것이다. 병렬 응용 프로그램을 SMP 환경에서 실행할 때 가장 심각한 문제중의 하나가 통신 지연에 관한 것이다. 따라서 본 논문에서 제안한 알고리즘은 연산비용보다 통신비용이 큰 환경에서 효과적

으로 사용될 수 있다.



(그림 6) *ccr*에 따른 스케줄링 길이의 변화(Systolic 알고리즘)

(그림 7)은 Master-Slave 알고리즘의 DAG를 사용하여 *ccr*의 변화에 따른 스케줄링 길이의 변화를 나타낸 것이다. 이 프로그램에서도 (그림 6)의 Systolic DAG와 유사한 결과가 나타났다. *ccr*이 증가하면 본 논문에서 제안한 스케줄링 알고리즘의 성능이 기존의 MSTES 알고리즘보다 성능이 우수하며, *ccr*의 값이 계속 증가하면 스케줄링의 길이가 급격히 증가한 한 후 변화가 없음을 확인할 수 있다.



(그림 7) *ccr*에 대한 스케줄링 길이의 변화(Master-slave 알고리즘)

5. 결 론

본 논문에서는 태스크 중복을 허용하고 휴리스틱을 사용하는 버스 기반의 SMP 환경에서 적용할 수 있는 태스크 스케줄링 알고리즘을 소개하였다. 알고리즘에서는 서로 다른 프로세서에 할당된 태스크 사이의 통신을 위하여 네트워크 통신 자원을 우선 배정하여 스케줄링 함으로써 네트워크 충돌을 방지한다. 또한 본 논문에서는 버스 기반의 SMP 환경에서 프로세서의 수, 연산비용에 대한 통신비용의 변화에 따른 스케줄링 길이의 변화를 분석하였다. 연산비용에 대한 통신비용의 비율이 높은 경우에 본 논문에서 제안한 알고리즘의 성능이 우수함을 시뮬레이션을 통하여 확인하였다.

참 고 문 헌

[1] G. F. Pfister, "In Search of Clusters," *Prentice Hall*, Englewood Cliffs, NJ, 1995.

[2] D. E. Culler et. al., "Parallel Computing on the Berkeley NOW," *9th Joint Symposium. on Parallel Processing*, Japan, 1997.

[3] F. M. Hayes, "Deaign of the AlphaServer Multiprocessor Server Systems," *Digital Technical Journal*, Vol.6, No.3, pp.8-19, 1994.

[4] M. Galles and E. Williams, "Performance Optimization, Implementation, and Verification of the SGI Challenge Multiprocessor," *Technical Report*. Silicon Graphics Computer Systems, CA, May, 1994.

[5] IBM Corporation, "RS/6000 SP System," *9th Joint Symposium*. IBM Corporation, RS/6000 Division, 1997.

[6] S. Nagar, A. Banerjee., A. Siva, and C.R. Das, "An Experimental Study of Scheduling Strategies for a Networks of Workstations," *Technical Report CSE-98-009*, Jul., 1997.

[7] X. Du and X. Zhang, "Coordinating Parallel Processes on Networks of Workstations," *Journal of Parallel and Distributed Computing*, Vol.46, pp.125-135, 1997.

[8] 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.

[9] 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.

[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] J. P. Kitajma and B. Plateau, "Building Synthetic Parallel Programs : The Project(ALPES)," *Pro. of IFIP WG 10.3 Workshop on Programming Environments for Parallel Computing*, pp.161-170, 1992.

강 오 한

e-mail : ohkang@andong.ac.kr

1982년 경북대학교 전자계열 전산모듈 졸업

1984년 한국과학기술원 전산학과 석사

1992년 한국과학기술원 전산학과 박사

1984년~1994년 (주) 큐닉스컴퓨터

1994년~현재 안동대학교 컴퓨터교육과 교수

관심분야 : 병렬처리, 스케줄링, WBI 등

김 시 관

e-mail : sgkim@se.kumoh.ac.kr

1982년 경북대학교 전자계열 전산모듈 졸업

1984년 한국과학기술원 전산학과 석사

2000년 한국과학기술원 전자전산학과 박사

1984년~1996년 삼성전자, LG정보통신 근무

2002년~현재 금오공과대학교 컴퓨터공학부 교수

관심분야 : 병렬처리, 이동통신, 스마트카드 등