태스크 복제 기반 프로세서 할당 방법에 최적화된 태스크 우선순위 결정 알고리즘☆

A Task Prioritizing Algorithm Optimized for Task Duplication Based Processor Allocation Method

송 인 성* 윤 완 오** 이 창 호*** 최 상 방****
Inseong Song Wanoh Yoon Changho Lee Sangbang Choi

요 약

분산 이기종 컴퓨팅 시스템의 성능은 입력 그래프인 방향성 비순환 그래프(DAG)를 스케줄링 하는 알고리즘의 성능에 따라 좌우된다. 그러나 분산 이기종 컴퓨팅 시스템에서의 태스크 스케줄링은 NP-complete 문제로 휴리스틱 방법으로 접근해야 한다. 태스크 스케줄링 알고리즘은 우선순위 결정 단계와 프로세서 할당 단계로 구성되며, 많은 연구들이 두 단계를 함께 고려하고 있다. 본 논문에서는 태스크 우선순위 결정 단계에 초점을 맞추어 태스크 복제 기반 프로세서 할당 방법에 최적화된 태스크 우선순위 결정 알고리즘인 WPD 알고리즘을 제안한다. 제안하는 WPD 알고리즘의 성능 분석을 위해 태스크 복제 기반 프로세서 할당 방법을 사용하는 기존의 태스크 스케줄링 알고리즘인 HMPID, HCPFD, HCT 알고리즘의 프로세서 할당 단계에 본 논문에서 제안하는 WPD 알고리즘을 결합하여 성능을 비교하였다. 그 결과 본 논문에서 제안하는 WPD 알고리즘이 기존 태스크 우선순위 결정 방법에 비해 태스크 복제를 더욱 효율적으로 사용하여 HCPFD 알고리즘보다 9.58%, HCT 알고리즘보다 1.31% 성능 향상이 있는 것을 확인하였다.

ABSTRACT

The performance of DHCS depends on the algorithm which schedules input DAG. However, as the task scheduling problem in DHCS is an NP-complete problem, heuristic approach has to be made. Task scheduling algorithm consists of task prioritizing phase and processor allocation phase, and most of studies are considering both phases together. In this paper, we focus on task prioritizing phase and propose a WPD algorithm which is optimized for task duplication based processor allocation method. For an evaluation of the proposed WPD algorithm, we combined WPD algorithm with processor allocation phase of HMPID, HCPFD, HCT algorithms, which are using task duplication based processor allocation method. The results show that WPD algorithm makes a better use of task duplication than conventional task prioritizing methods and provides 9.58% better performance than HCPFD algorithm, 1.31% than HCT algorithm.

☞ keyword : DHCS(분산 시스템), task scheduling(태스크 스케줄링), task duplication(태스크 복제), DAG(방향성 비순환 그래프)

1. 서 론

네트워크 기술의 발달로 서로 다른 성능을 갖는 프로세서들 간에 효율적인 통신이 가능하게 되면서 다량의 데이터베이스를 저장하고 고속으로 처리할 수 있는 분산 시스템이 주목받고 있다. 그 중에서도 고속의 네트워크를 이용하여 서로 다른 성능을 갖는 프로세서를 연결한 분산 이기종 컴퓨팅 시스템(Distributed Heterogeneous Computing System, DHCS)의 연구가 활발히 이루어지고 있다.

* 정 회 원 : 인하대학교 대학원 전자공학과 박사과정
nicvirus@inha.edu (교신저자)
** 정 회 원 : 인하대학교 정보전자공동연구소 연구교수
wanoh38@paran.com
*** 정 회 원 : 인하대학교 대학원 전자공학과 박사과정
lchh0902@nate.com
**** 종신회원 : 인하대학교 전자공학과 교수
sangbang@inha.ac.kr
[2011/04/04 투고 - 2011/04/17 심사(2011/07/26 2차) - 2011/08/26 심사완료]
☆ 이 논문은 정부(교육과학기술부)의 재원으로 한국연구재단의 중점연구소 지원사업으로 수행된 연구임(2011-0018394)

분산 이기종 컴퓨팅 시스템에서 실행되는 병렬 프로그램은 방향성 비순환 그래프(Directed Acyclic Graph, DAG)로 모델링할 수 있다. DAG는 처리해야할 작업을 나타내는 태스크와, 다른 태스크와의 데이터 의존도를 나타내는 에지를 포함한다. 분산 이기종 컴퓨팅 시스템이 병렬 프로그램을 처리하는 성능은 입력 DAG를 각 프로세서에 할당하는 태스크 스케줄링 알고리즘의 성능에 따라 좌우된다. 태스크 스케줄링의 목적은 입력 DAG의 특성에 따라 최적의 프로세서에 태스크를 할당하여 병렬 프로그램의 전체실행시간을 최소화하는 것이다. 그러나 이는 NP-complete 문제로 많은 연구들이 허용 가능한 시간 내에서 최적에 가까운 해를 찾는데 중점을 두고있다[1-3].

본 논문에서는 태스크 복제 기반 프로세서 할당 방법에 최적화된 새로운 우선순위 결정 알고리즘인 WPD(Weight based task Prioritizing for Duplication) 알고리즘을 제안한다. WPD 알고리즘은 두 단계로 이루어진다. 첫 번째 단계인 계층화 단계에서는 입력 DAG 내의 태스크를 서로 독립인 태스크들끼리 묶어 계층을 만든다. 두 번째 단계인 우선순위 계산 단계에서는 태스크의 평균계산비용에 대한 내림차순으로 정렬하여 계층 내에서의 우선순위를 결정하고, 최종적으로 전체 우선순위를 결정한다.

공정한 성능 평가를 위해 기존 연구에서 제안한 표준 태스크 그래프와 실제 응용프로그램을 모델링한 그래프를 시뮬레이션의 입력 DAG로 사용하였다. 성능 평가에는 태스크 복제 기반 프로세서 할당 방법을 사용하는 기존 알고리즘인 HMPID, HCPFD, HCT 알고리즘과, 본 논문에서 제안하는 WPD 알고리즘에 이들 기존 알고리즘의 프로세서 할당 단계를 결합한 알고리즘을 이용하였다. 실험결과 WPD 알고리즘을 프로세서 할당 과정에서 태스크 복제를 사용하는 HCPFD, HCT 알고리즘과 결합했을 때 HCPFD 알고리즘보다 평균 약 9.58%, HCT 알고리즘보다 평균 약 1.31% 향상된 성능을 보였다. 하지만 태스크의 삽입과 복제를 함께 사용하는 HMPID 알고리즘과 결합했을 때 약간의 성능 하락이 있었다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 분산 이기종 컴퓨팅 시스템 환경과 DAG에 관해 설명하며, 3장에서는 기존의 대표적인 스케줄링 알고리즘에 관해 설명한다. 4장에서는 본 논문에서 제안하는 WPD 알고리즘을 소개한다. 5장에서는 본 논문에서 제안하는 알고리즘과 기존의 알고리즘에 대한 성능 실험 및 평가를 보이고, 마지막으로 6장에서는 본 논문의 연구 내용에 대한 최종 결론을 정리한다.

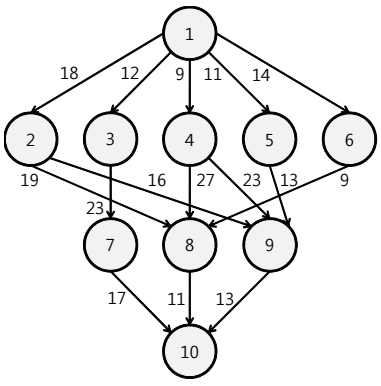
2. 스케줄링을 위한 문제 정의

분산 이기종 컴퓨팅 시스템은 m개의 이기종 프로세서 P가 완전연결로 구성된 시스템을 뜻한다. 각 프로세서 사이의 통신채널 및 대역폭은 충분하다고 가정하며, 각 프로세서는 태스크 실행과 데이터 통신을 동시에 처리할 수 있다고 가정한다. 또한 프로세서는 태스크를 처리하는 동안 어떠한 방해도 받지 않고 태스크의 실행에만 집중할 수 있으며, 실행이 완료된 결과는 지연 없이 다른 프로세서로 전달할 수 있다고 가정한다.

분산 이기종 컴퓨팅 시스템에서 실행되는 병렬 프로그램은 비순환 방향성 그래프 DAG, $G=(V,E)$로 모델링할 수 있다. 일반적으로 DAG는 (그림 1) (a)와 같이 표현하며, 원으로 표현되는 태스크와 화살표로 표현되는 에지로 이루어져있다. 태스크는 처리해야 할 작업을 의미하며, 에지는 태스크의 선후관계를 의미한다. 각 프로세서에서 실행되는 태스크의 계산비용은 (그림1) (b)와 같이 별도로 주어지게 된다.

프로세서집합 $P=\{P_1, P_2, ..., P_m\}$은 분산 이기종 컴퓨팅 시스템 환경의 모든 프로세서를 포함한다. DAG $G=(V,E)$에서 태스크집합 $V=\{v_1, v_2, ..., v_{n-1}, v_n\}$는 DAG 내의 모든 태스크를 포함하며, i번째 태스크는 v_i로 표현한다.

에지집합 $E=\{e_{1,2}, e_{1,3}, ..., e_{i,j}\}$는 각 태스크를 연결하는 에지들을 포함한다. 하나의 에지는



(a)

Task	P_1	P_2	P_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

(b)

(그림 1) (a) 입력 DAG, (b) 프로세서 별 태스크 실행 시간

두 개의 태스크를 연결하며 태스크 간의 선후관계를 나타낼 수 있다. 예를 들어, 에지 $e_{i,j}$는 태스크 v_i와 태스크 v_j를 연결하고, 이는 태스크 v_i의 실행이 종료되고 난 후 태스크 v_j가 실행될 수 있다는 의미를 갖는다. 이 때, 태스크 v_i는 태스크 v_j의 부모태스크가 되고, 태스크 v_j는 태스크 v_i의 자식태스크가 된다. 또한 태스크 v_i의 모든 자식태스크의 집합을 $succ(v_i)$, 태스크 v_j의 모든 부모태스크의 집합을 $pred(v_j)$로 정의한다. 마지막태스크는 자식태스크가 존재하지 않으며, 마지막태스크를 제외한 다른 모든 태스크는 하나 이상의 자식태스크를 갖는다. 또한 시작태스크는 부모태스크가 존재하지 않으며, 시작태스크를 제외한 다른 모든 태스크는 하나 이상의 부모태스크를 갖는다. 본 논문에서 입력으로 주어지는 DAG는 반드시 하나의 시작태스크와 마지막태스크를 가져야 한다. 만일 2개 이상의 시작태스크와 마지막태스크가 존재한다면 이 태스크를 묶을 수 있는 더미태스크를 추가할 수 있으며, 더미태스크는 계산비용과 통신비용이 모두 0이므로 스케줄링에 아무런 영향을 주지 않는다. 부모태스크 집합 $pred(v_i)$에 포함된 태스크 중, 태스크 v_i에 데이터를 가장 늦게 전달하는 부모태스크는 임계부모태스크 $dpred(v_i)$로 정의한다.

평균통신비용 $\bar{c}(v_i, v_j)$는 태스크 v_i에서 태스크 v_j로 데이터를 전달하는데 걸리는 평균 시간을 의미한다. 평균통신비용은 에지 상의 가중치로 표현되며, 두 태스크가 동일한 프로세서에서 실행된다면 0의 값을 갖는다. 계산비용 $w(v_i, P_m)$은 프로세서 P_m에서 태스크 v_i가 실행되는데 소요되는 시간을 의미한다. 평균계산비용 $\overline{w_i}$는 식 (1)과 같이 모든 프로세서에 대한 태스크 v_i의 평균계산비용을 의미한다.

$$\overline{w_i} = \sum_{m=1}^{n} \frac{w(v_i, P_m)}{n} \quad (1)$$

태스크 v_i부터 마지막태스크 v_n까지 가장 긴 경로를 $b_level(v_i)$로 정의하고 식 (2)를 이용하여 마지막태스크부터 재귀적으로 계산한다. 마지막태스크의 $b_level(v_n)$은 $\overline{w_n}$로 정의된다. 마찬가지로 특정 태스크 v_i부터 시작태스크 v_1까지 가장 긴 경로를 $u_level(v_i)$로 정의하고 식 (3)을 이용하여 시작태스크부터 재귀적으로 계산한다. 임계경로는 시작태스크부터 마지막태스크까지 가장 긴 경로를 의미한다[4].

$$b\_level(v_i) = \overline{w}(v_i) + \max_{v_j \in succ(v_i)} \{\bar{c}(v_i, v_j) + b\_level(v_j)\} \quad (2)$$

$$u\_level(v_i) = \max_{v_j \in pred(v_i)} \{u\_level(v_j) + \overline{w_j} + \bar{c}(v_i, v_j)\} \quad (3)$$

프로세서가용시간 $pat(P_j)$는 프로세서 P_j에서 태스크가 실행을 시작할 수 있는 시간을 의미한다. 즉, 프로세서 P_j에 할당된 태스크 중 가장 늦게 실행이 종료되는 태스크의 실행완료시간이 프로세서 P_j의 프로세서가용시간이 된다. 최소실행시작시간 $est(v_i, P_m)$은 태스크 v_i가 모든 부모태스크로부터 데이터를 전달받았으며 프로세서가 사용 가능할 때 프로세서 P_m에서 실행이 가능한 시간을 의미하며, 식 (4)와 같이 정의된다. 태스크 v_i가 프로세서 P_m에서 실행이 완료되는 최소실행완료시간 $eft(v_i, P_m)$은 식 (5)와 같이 정의된다[4].

$$est(v_i, P_m) = \begin{cases} 0 & if\ v_i = v_1 \\ \max\limits_{v_j \in pred(v_i)} \{pat(P_m), \\ \quad \max(eft(v_j) + \bar{c}(v_i, v_j))\} & otherwise \end{cases} \quad (4)$$

$$eft(v_i, P_m) = est(v_i, P_m) + w(v_i, P_m) \quad (5)$$

Makespan은 입력 DAG를 분산 이기종 컴퓨팅 시스템에서 실행하는데 소요되는 시간을 의미한다. 따라서 Makespan은 다음 식 (6)과 같이 마지막태스크 v_n의 P_m에서 최소실행완료시간 $eft(v_n, P_m)$으로 정의한다.

$$Makespan = eft(v_n, P_m) \quad (6)$$

3. 관련 연구

태스크 스케줄링에 관한 연구가 진행되고 있는 분야는 태스크 스케줄링을 실행하는 시점에 따라 동적 스케줄링과 정적 스케줄링으로 나눌 수 있으며, 정적 스케줄링은 크게 휴리스틱 기반 스케줄링과, 임의 탐색 기반 스케줄링으로 나누어진다. 휴리스틱 기반 스케줄링은 다시 리스트 스케줄링[4-6]과 태스크 복제 기반 스케줄링[7-11], 클러스터 스케줄링으로 분류 할 수 있다. 리스트 스케줄링에 속하는 알고리즘은 HEFT[4], PETS[5], HRPS[6] 등이 있으며, 태스크 복제 기반 스케줄링에 속하는 알고리즘은 HCPFD[8], DCPD[9], HCT[10], HEFD[11] 등이 있다. 태스크 복제 기반 스케줄링은 태스크 간의 선행 제약에 의한 통신비용을 줄이기 위해 선행 태스크의 복제를 사용하며, 시간 복잡도가 높다는 단점이 있지만 우수한 스케줄링 결과를 제공한다.

이 장에서는 본 논문에서 제안하는 알고리즘과 관련이 있는 대표적인 태스크 스케줄링 알고리즘에 대해 설명한다.

3.1 HMPID 스케줄링 알고리즘

HMPID(Heterogeneous Multi Processor system considering Insertion and Duplication)[7] 알고리즘은 태스크의 삽입과 복제를 동시에 고려하는 스케줄링 알고리즘이다. 우선순위 결정 단계에서는 입력 DAG내에 있는 모든 태스크의 $b_level(v_i)$을 계산하고 $b\_level(v_i)$ 값에 대한 내림차순으로 정렬하여 우선순위를 결정한다.

프로세서 할당 단계에서는 결정된 우선순위가 가장 높은 태스크를 선택하여 가장 작은 최소실행완료시간을 제공하는 프로세서에 태스크를 할당한다. 이 때 태스크의 삽입을 이용하여 현재 태스크의 최소실행완료시간을 줄일 수 있다면 현재 태스크를 빈 공간에 삽입한다. 삽입이 발생하지 않는다면 현재 태스크의 임계부모태스크를 복제하여 최소실행완료시간을 줄일 수 있는지 확인하고 임계부모태스크를 복제한다. HMPID 알고리즘의 시간복잡도는 $O(v^2p)$이다.

HMPID 알고리즘은 태스크의 삽입과 복제를 함께 사용하여 뛰어난 성능을 제공하고, WPD 알고리즘을 태스크 삽입을 사용하는 프로세서 할당 단계와 결합했을 때의 영향을 알아볼 수 있기 때문에 본 논문의 성능 평가에 사용되었다.

3.2 HCPFD 스케줄링 알고리즘

HCPFD(Heterogeneous Critical Parents with Fast Duplicator)[8] 알고리즘은 태스크 복제 기반 스케줄링 알고리즘이다. 태스크 우선순위 결정 단계에서는 입력 DAG의 마지막태스크부터 시작하여 상향으로 탐색하며 모든 태스크의 최대평균시작시간과 최소평균시작시간을 계산하고, 두 값의 차가 0인 태스크들을 선택해 임계경로를 생성한다. 다음으로 임계경로 상에 있는 태스크를 스택에 푸시한다. 스택의 최상위에 있는 태스크에 대하여 우선순위 목록에 태스크의 부모태스크가 존재하는지 확인하고, 부모태스크가 큐에 존재하지 않는다면 부모태스크를 스택에 푸시한다. 부모태스크가 큐에 존재한다면 태스크를 팝하여 큐에 저장한다. 이 과정은 스택에 남는 태스크가 없을 때까지 계속된다. 우선순위 큐 목록이 최종 태스크 우선순위 목록이 된다.

프로세서 할당 단계에서는 우선순위 목록의 첫 번째 태스크를 선택하여 모든 프로세서에 대해 최소실행완료시간을 계산하고, 현재 태스크가 할당된 프로세서에 임계부모태스크를 복제할 수 있는지 확인한다. 만일 임계부모태스크 복제를 통해 현재 태스크의 최소실행시간을 앞당길 수 있다면 임계부모태스크의 복제가 이루어지며, 리스트에 있는 모든 태스크가 프로세서에 할당될 때까지 이 과정이 반복된다. HCPFD 알고리즘의 시간 복잡도는 $O(v^2p)$이다.

HCPFD 알고리즘은 프로세서 할당 단계에서 임계부모태스크의 복제를 활용하는 대표적인 알고리즘이기 때문에 본 논문의 성능 평가에 사용되었다.

3.3 DCPD 스케줄링 알고리즘

DCPD(Dynamic Critical Path Duplication)[9] 알고리즘은 태스크 복제 기반 스케줄링 알고리즘이다. 우선순위 결정 단계에서는 입력 DAG의 마지막태스크부터 시작하여 상향으로 탐색하며 모든 태스크의 $b_level(v_i)$를 계산한다. 다음으로 모든 부모태스크의 실행이 완료된 태스크를 준비집합에 넣고, $b\_level(v_i)$, $u\_level(v_i)$, $\overline{w_i}$ 값을 이용하여 준비집합 내에 있는 태스크들의 우선순위를 결정한다.

프로세서 할당 단계에서는 결정된 우선순위대로 각 프로세서에서의 최소실행완료시간을 계산하여 가장 작은 최소실행완료시간을 제공하는 프로세서에 태스크를 할당한다. 할당 과정에서 복제가 고려되며, 준비집합 내의 모든 태스크가 프로세서에 할당될 때 까지 이 작업이 반복된다. DCPD 알고리즘의 시간 복잡도는 $O((v+e)vp)$이다.

4. 제안하는 스케줄링 알고리즘

이 장에서는 본 논문에서 제안하는 WPD 알고리즘에 대해 설명한다.

4.1 WPD 알고리즘

본 논문에서는 복제 기반 태스크 할당 방법에 최적화된 태스크 우선순위 결정 단계 알고리즘인 WPD(Weight based task Prioritization for Duplication) 알고리즘을 제안한다. 일반적으로 태스크 스케줄링 알고리즘은 태스크 우선순위 결정 단계와 프로세서 할당 단계의 두 단계로 이루어지며, 본 논문에서 제안하는 WPD 알고리즘은 우수한 스케줄링 성능을 제공하는 태스크 복제 기반 프로세서 할당 방법을 보다 더 효율적으로 사용하기 위해 태스크 우선순위 결정 단계에 초점을 맞춘 알고리즘이다.

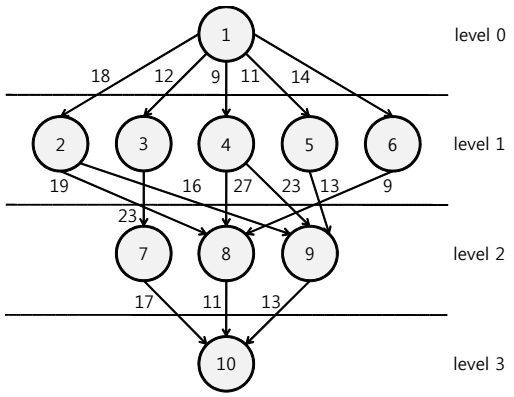
WPD 알고리즘의 특징은 입력 DAG의 계층화와 태스크의 계산비용만을 이용한 우선순위 결정이다. 입력 DAG의 계층화는 서로 독립적인 태스크들이 동시에 실행될 수 있게끔 하여 병렬성을 높이기 위한 것이며, 계산비용을 이용하여 태스크의 우선순위를 정하는 것은 계산비용이 큰 태스크일수록 전체실행시간에 미치는 영향이 크기 때문에 계산비용이 큰 태스크에 프로세서 선택의 우선권을 주기 위한 것이다. 우선순위 결정 과정에서 기존의 태스크 우선순위 결정 알고리즘들과 달리 통신비용을 고려하지 않으면서 발생하는 통신대기시간 문제는 태스크 복제를 통해 전체 프로세서에서의 스케줄

링 문제를 한 프로세서 내에서의 지역적 스케줄링 문제로 바꿈으로써 해결할 수 있다.

WPD 알고리즘은 세부적으로 계층 정렬 단계와 우선순위 계산 단계의 두 단계로 나누어진다. 첫 번째 단계인 계층 정렬 단계에서는 DAG 상의 태스크들을 계층별로 분류한다. 두 번째 단계인 우선순위 계산 단계에서는 각 태스크의 평균 계산 비용에 따라 각 계층 내에서 태스크들의 우선순위를 결정하고, 이를 종합한 뒤, 프로세서 할당을 위한 최종 우선순위 목록을 작성한다. 프로세서 할당 단계에서는 WPD 알고리즘을 통해 얻어진 우선순위를 바탕으로 태스크 복제를 고려한 프로세서 할당이 이루어진다. 다시 말해서, WPD 알고리즘은 DAG의 계층화와 계산비용을 이용한 우선순위 결정으로 태스크의 복제를 최대한 활용하여 병렬 프로그램의 실행 성능을 향상시키는 새로운 태스크 우선순위 결정 알고리즘이다.

첫 번째 단계인 계층 정렬 단계에서는 입력 DAG 상의 모든 태스크들이 계층별로 분류된다. 계층 정렬을 위해 DAG 상의 모든 태스크를 시작태스크부터 마지막태스크까지 너비 우선 검색(Breadth First Search, BFS)를 이용하여 독립적인 태스크의 집합으로 계층화한다. 한 계층은 독립적인 태스크들의 집합이므로 같은 계층에 속한 태스크들 사이에는 선행제약이 없으며 동시에 실행이 가능하다. i 번째 계층에는 모든 에지 $e(v_j, v_k)$에 대하여 $i-1$ 번째 계층에 있는 v_j와 적어도 하나 이상의 에지 $e(v_j, v_k)$로 연결된 태스크 v_k가 포함된다. 이 때, 0 번째 계층은 시작태스크를 포함하며 마지막태스크는 마지막 계층에 속한다.

(그림 1) (a)의 입력 DAG를 계층화 한 결과는 (그림 2)와 같다. 그림 내에서 각 계층은 점선으로 구분되어 있다. 1번 태스크 v_1은 시작태스크이므로 첫 번째 계층인 L_0에 포함되어 $level(v_1) = L_0$가 된다. 이어서 계층 L_0에 속한 태스크 v_1과 에지 $e(v_1, v_2)$로 연결된 v_2는 두 번째 계층인 L_1에 포함되어 $level(v_2) = L_1$이 된다. 마찬가지로 태스크 v_3, v_4, v_5, v_6는 모두 두 번째 계층인 L_1에 포



(그림 2) DAG 계층화의 예

함된다. 다음으로 계층 L_1에 속한 태스크 v_3와 에지 $e(v_3, v_7)$로 연결된 v_7은 세 번째 계층인 L_2에 포함된다. 태스크 v_8은 태스크 v_2, v_4, v_6와 에지 $e(v_2, v_8)$, $e(v_4, v_8)$, $e(v_6, v_8)$로 연결되었으므로 L_2에 포함되며, 마찬가지로 태스크 v_9 역시 L_2에 포함된다. 마지막으로 마지막태스크인 v_{10}은 마지막 계층인 L_3에 포함된다. 결과적으로 (그림 1) (a)의 DAG는 $L_0 = \{v_1\}$, $L_1 = \{v_2, v_3, v_4, v_5, v_6\}$, $L_2 = \{v_7, v_8, v_9\}$, $L_3 = \{v_{10}\}$으로 계층화 된다.

두 번째 단계인 우선순위 계산 단계에서는 각 계층에 있는 태스크들의 우선순위를 결정하고 최종 우선순위 목록을 작성한다. WPD 알고리즘은 태스크의 우선순위 결정 시 태스크의 평균계산비용을 이용하여 우선순위를 결정한다. 이는 계산비용이 큰 태스크일수록 스케줄링 결과에 미치는 영향이 크기 때문에 우선적으로 최적의 프로세서에 할당하여 최적의 스케줄을 얻기 위함이다. 태스크의 평균계산비용은 식 (1)을 이용하여 구할 수 있다. 우선 각 계층 내에 있는 모든 태스크들의 평균계산비용을 구한 뒤 평균계산비용에 대한 내림차순으로 태스크들을 정렬한다. 각 계층 내에서 태스크의 우선순위를 정하는 과정을 마치면 낮은 계층부터 높은 계층까지, 즉 첫 번째 계층부터 마지막 계층까지 각 계층 내의 태스크 우선순위를 차례대로 나열

(표 1) 각 태스크의 평균계산비용과, 계층 내에서의 우선순위, 및 총 우선순위

Level	Task	$\overline{w_i}$	Priority in level	Final priority
0	1	13	1	1
1	2	16.67	1	2
	3	14.33	2	3
	4	12.67	3	4
	5	11.67	5	6
	6	12.67	4	5
2	7	11	2	8
	8	10	3	9
	9	16.67	1	7
3	10	14.67	1	10

하여 최종 태스크 우선순위 목록을 작성한다.

예를 들어 (그림 1)과 같이 입력 DAG와, 3개의 프로세서에서 각 태스크의 계산 비용이 주어졌을 때 각 태스크의 평균계산비용과 계층 내에서의 우선순위, 최종 우선순위는 (표 1)과 같이 계산된다. 표 내에서 Level은 각 태스크가 속하는 계층을, Task는 각 태스크의 번호를, $\overline{w_i}$는 평균계산비용을, Priority in level은 계층 내에서 태스크의 우선순위를, Final priority는 태스크의 최종 우선순위를 뜻한다.

먼저 첫 번째 계층인 L_0에 속하는 태스크 v_1은 프로세서 P_1에서 14, 프로세서 P_2에서 16, 프로세서 P_3에서 9의 계산비용을 가지며 시스템에는 총 3개의 프로세서가 존재한다. 따라서 $\overline{w_1}=(w_{1,1}+w_{1,2}+w_{1,3})/n$ 이므로 태스크 v_1은 $\overline{w_1}=(14+16+9)/3=13$의 평균계산비용을 갖게 된다. 계층 L_0 내에는 단 하나의 태스크만 존재하므로 평균계산비용에 따른 내림차순 정렬이 필요 없으며 태스크 v_1의 계층 L_0 내에서의 우선순위는 1번으로 계층 L_0의 우선순위 목록은 $\{v_1\}$이 된다.

다음으로 두 번째 계층인 L_1에 속한 태스크 v_2는 프로세서 P_1에서 13, 프로세서 P_2에서 19, 프로세서 P_3에서 18의 계산비용을 갖는다. 따라서 태스크 v_2의 평균계산비용은 $\overline{w_2}=(13+19+18)/3=16.67$을 갖게 된다. 같은 방법으로 계층 L_1에 속한 나머지 태스크 v_3, v_4, v_5, v_6는 $\overline{w_3}=14.33$, $\overline{w_4}=12.67$, $\overline{w_5}=11.67$, $\overline{w_6}=12.67$이 된다. 계층 L_1 내에서의 우선순위는 계산된 평균계산비용을 내림차순으로 정렬하여 결정된다. 제일 먼저 16.67로 가장 큰 평균계산비용을 갖는 태스크 v_2가 나열되고, 다음으로 14.33의 평균계산비용을 갖는 태스크 v_3가 나열된다. 태스크 v_4와 태스크 v_6는 12.67로 같은 평균계산비용을 갖는다. 일반적으로 분산 이기종 컴퓨팅 시스템 환경에서 실행되는 병렬 프로그램을모델링한 DAG의 같은 계층 내에 속한 태스크들이 동일한 평균계산비용을 갖는 경우가 많을 것이라고 기대하기 어렵기 때문에 이와 같은 경우 복잡도를 낮추기 위하여 임의로 우선순위를 설정하게 된다. 따라서 태스크 v_4, v_6의 순서로 우선순위를 결정하고, 마지막으로 11.67로 가장 작은 평균계산비용을 갖는 태스크 v_5를 계층 내 우선순위 목록에 포함시킨다. 따라서 계층 L_1 내의 우선순위 목록은 $\{v_2, v_3, v_4, v_6, v_5\}$가 된다.

세 번째 계층 L_2에 속한 태스크의 평균계산비용은 $\overline{w_7}=11$, $\overline{w_8}=10$, $\overline{w_9}=16.67$이다. 마찬가지로 계층 L_2 내에서의 우선순위는 평균계산비용의 크기에 따라 태스크 v_9을 가장 먼저 우선순위 목록에 포함시키고, 태스크 v_7, v_8을 차례로 우선순위 목록에 포함시키면 최종적으로 계층 L_2의 우선순위 목록은 $\{v_9, v_7, v_8\}$이 된다.

마지막 계층 L_3에 속한 태스크 v_{10}은 $\overline{w_{10}}=14.67$의 평균계산비용을 갖는다. 계층 L_3 역시 계층 L_0와 마찬가지로 계층 내에 단 하나의 태스크만 존재하므로 평균계산비용에 따른 내림차순 정렬이 필요 없으며 태스크 v_{10}의 계층 L_3 내에서의 우선순위는 1번으로 계층 L_3의 우선순위 목록은 $\{v_{10}\}$이 된다.

각 계층 내에서 태스크의 우선순위를 정하는 과정이 완료되면 첫 번째 계층 L_0부터 마지막 계층

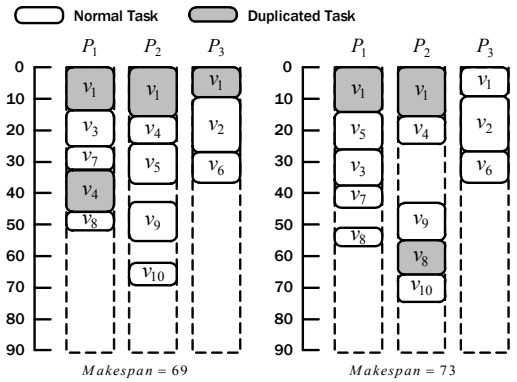
```
/* Level sorting stage */
Find the number of levels in DAG
// Count tasks in a level
for all k tasks do
        if level(v_k) ≠ L_i
                increase i
        else
                increase taskcount
endfor
// Create level groups
Create i level groups
// Add tasks to level groups
for all k tasks do
        add v_k to corresponding level group L_i
endfor

/* Task priority computation stage */
for all i levels do
        // Compute average weights
        for each task in each level group L_i do
                Compute w_i of the task
        endfor
        // Make partial ordered lists
        Queue tasks in decreasing order of w_i
endfor
// Make final priority list
Make a final priority list for input DAG
```

(그림 3) WPD 알고리즘의 의사 코드

L_3까지 각 계층 내의 태스크 우선순위를 차례대로 나열하여 최종 태스크 우선순위 목록을 작성한다. 따라서 최종 태스크 우선순위 목록은 $\{v_1, v_2, v_3, v_4, v_6, v_5, v_9, v_7, v_8, v_{10}\}$이 된다.

(그림 3)은 WPD 알고리즘의 의사 코드이다. 가장 먼저 계층 정렬을 위해 DAG 내에 있는 계층의 개수를 찾은 뒤 각 계층에 포함되는 태스크의 개수를 계산한다. 다음으로 계층 집합을 DAG 내의 계층 개수만큼 생성하고 모든 태스크를 차례로 검사해가며 각 태스크를 해당하는 계층 집합에 포함시킨다. 계층 정렬이 완료되면 우선순위 결정을 위해 각 계층 내에서 모든 태스크들에 대하여 평균계산비용을 계산하고 계산비용의 내림차순으로 정렬한다. 모든 계층의 내부 우선순위 계산이 완료되면 마지막으로 최종 우선순위 목록을 작성한다.



(a) WPD(HCPFD) (b) HCPFD

(그림 4) 스케줄링 결과

(그림 4) (a)는 (그림 1) (a)의 입력 DAG에 WPD 알고리즘을 적용하여 생성된 우선순위 목록에, 태스크 복제를 사용하는 HCPFD 알고리즘의 프로세서 할당 단계를 적용하여 생성된 스케줄링 결과이다. (그림 4) (b)는 (그림 1) (a)의 입력 DAG에 HCPFD 알고리즘을 적용하여 생성된 스케줄링 결과이다. 이 예에서 사용된 HCPFD 알고리즘의 프로세서 할당 단계는 태스크를 프로세서에 할당하기 위해 먼저 태스크 우선순위 목록의 첫 번째 태스크를 v_i로 정의하고 모든 프로세서 P_j에 대하여 태스크 v_i의 최소실행완료시간 $eft(v_i, P_j)$를 계산한다. 다음으로 태스크 v_i의 최소실행완료시간을 제공하는 프로세서 P_m을 선택하고 태스크 v_i의 임계부모태스크를 찾아 v_{cp}로 정의한다. 이 때 프로세서 P_m에 임계부모태스크 v_{cp}를 복제할 수 있는 공간이 충분하다면 임계부모태스크 v_{cp}의 복제를 실행한다. 이 과정은 우선순위 목록에 더 이상 태스크가 남지 않을 때까지 반복된다. 할당 결과 WPD 알고리즘과 HCPFD 알고리즘을 결합한 알고리즘을 이용한 경우 전체실행시간 Makespan은 69이며, HCPFD 알고리즘을 이용한 경우 전체실행시간 Makespan은 73이다. 그림에서 음영으로 표시된 부분은 복제된 태스크를 의미한다.

4.2 제안하는 알고리즘의 복잡도

알고리즘의 시간 복잡도는 알고리즘을 구성하는 명령어들이 실행된 횟수와 각 명령어의 실행시간을 곱한 합계를 의미한다. 그러나 각 명령어의 실행시간은 하드웨어와 프로그래밍 언어에 종속적이므로 상황에 따라 그 값이 달라질 수 있다. 따라서 알고리즘의 일반적인 시간 복잡도는 명령어의 실제 실행시간을 고려하지 않은 명령어의 실행 횟수만을 고려한다.

WPD 알고리즘의 시간 복잡도는 (그림 3)의 의사코드를 통해 알아볼 수 있다. 우선 DAG를 계층화하는 계층 정렬 단계에서 너비 우선 검색을 사용하면서 $O(v+e)$의 시간이 소요된다. 다음으로 우선순위 계산 단계에서 우선순위 목록을 저장하기 위한 큐로 바이너리 힙이 사용되므로 여기에서 $O(\log v)$의 시간이 소요된다. 따라서 WPD 알고리즘은 최종적으로 $O(v+e)(\log v)$의 시간 복잡도를 갖는다.

5. 성능 분석 및 평가

이번 장에서는 본 논문에서 제안하는 WPD 알고리즘의 성능 분석을 위해 HMPID, HCPFD, HCT 알고리즘과 성능을 비교한다. 본 논문에서는 기존의 연구에서 제안한 표준 태스크 그래프와 실제 응용프로그램으로부터 생성된 그래프에 입력 DAG에 포함된 태스크의 수, 통신비용 대 계산비용, 프로세서 이질성도, 프로세서 개수를 매개변수로 적용하여 실험하였다.

5.1 성능 비교 기준

공정하고 정확한 알고리즘의 성능 비교를 위하여 기존 논문에서 사용되었던 다양한 비교 기준들 중에서 다음의 두 기준을 이용하여 각 알고리즘의 성능을 비교하였다.

• SLR

Schedule Length Ratio(SLR)은 Makespan을 생성되는 스케줄의 하한 값으로 표준화한 값이다. 성능 비교에는 각각 다른 속성을 가진 많은 수의 입력 DAG가 이용되기 때문에 Makespan보다 공정한 비교 기준이라 할 수 있다. SLR은 다음 식 (7)과 같이 정의된다.

$$SLR = \frac{Makespan}{\sum_{v_i \in CP} min_{p_m \in P}\{w_{i,m}\}} \quad (7)$$

식 (7)의 분모는 입력 DAG의 임계경로 상에 존재하는 태스크들의 최소계산비용의 합이다. 하한 값을 분모로 이용하기 때문에 어떠한 알고리즘을 사용하더라도 알고리즘의 SLR은 1보다 작아질 수 없다. 따라서 가장 작은 SLR을 나타내는 알고리즘이 좋은 성능을 제공하는 알고리즘이라고 할 수 있다.

• Speedup

Speedup은 순차 실행 시간을 병렬 실행 시간으로 나눈 값이다. 순차 실행 시간은 모든 태스크를 계산비용의 합이 최소가 되는 단일 프로세서에 할당했을 때의 Makespan이고, 병렬 실행 시간은 입력 DAG를 스케줄링 알고리즘을 이용하여 병렬 실행했을 때의 Makespan이다. Speedup은 식 (8)로 정의된다.

$$Speedup = \frac{min_{p_m \in P}\{\sum_{v_i \in V} w_{i,m}\}}{makespan} \quad (8)$$

5.2 입력 매개변수 및 입력 DAG

알고리즘의 공정한 성능 평가를 위해서 표준 태스크 그래프 프로젝트(STanDard task Graph Project, STDGP)[12]에서 제공하는 DAG와 실제 응용프로그램으로부터 생성된 DAG를 이용하였으며, 이러한 DAG에 매개변수를 적용하여 실험하였다.

• DAG의 태스크 개수(v)

입력 DAG에 포함된 태스크의 개수를 의미한다. 본 논문에서는 $\{50, 100, 300, 500\}$개의 태스크

를 갖는 DAG를 각 태스크 개수 별로 120개씩, 총 480개의 입력 DAG를 이용하여 실험하였다.

• 프로세서 개수

성능 평가를 위해 다양한 경우의 분산 이기종 컴퓨팅 시스템 환경을 가정하였다. 본 논문에서는 최소 2개에서 최대 32개까지, $\{2, 4, 8, 16, 32\}$ 개의 프로세서를 사용하여 실험하였다.

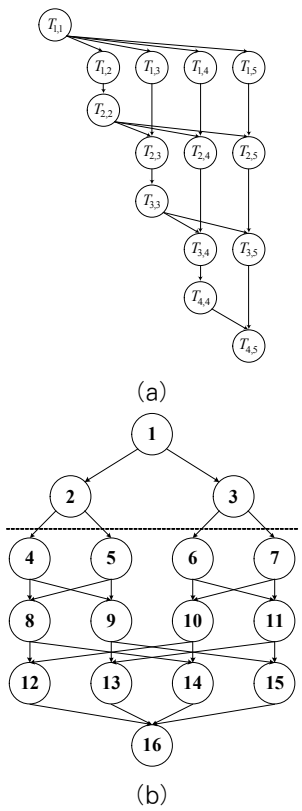
• 통신비용 대 계산비용 비(CCR)

통신비용 대 계산비용의 비율(Communication to Computation Ratio, CCR)은 평균 통신비용에 대한 평균 계산비용의 비율이다. 만일 통신비용 대 계산비용의 비율이 1보다 작다면 이는 데이터의 연산에 전달보다 많은 시간을 소모하는 연산 집약적인 응용 프로그램이라 할 수 있으며, 통신비용 대 계산비용의 비율이 1보다 크다면 이는 데이터의 전달에 연산보다 더 많은 시간을 소모한다는 것이다. 본 논문에서는 $\{0.1, 0.5, 1.0, 2.5, 5.0\}$ 의 값을 이용하여 실험하였다.

• 프로세서의 이질성도(β)

분산 이기종 시스템을 구성하는 각 프로세서 간의 성능 차이를 나타낸다. 프로세서 이질성도를 통해 각 태스크의 프로세서 별 실행시간을 결정한다. 프로세서 이질성도가 클수록 어떤 태스크를 실행할 때 프로세서 별 실행시간의 차이가 크고, 작을수록 프로세서 별 태스크 실행시간의 차이가 거의 없어진다. DAG 내의 각 태스크 v_i의 평균계산비용 $\overline{w_i}$는 균일 분포를 갖는 $[0, 2\times\overline{w}_{DAG}]$ 범위 내에서 임의로 선택된다. 여기에서 $\overline{w}_{DAG}$는 주어진 DAG의 계산비용의 평균값이다. 따라서 각 태스크 v_i의 평균계산비용은 다음 식 (9)의 범위 내에 속하게 된다. 본 논문에서는 $\{0.1, 0.5, 1.0, 1.5, 2.0\}$ 의 값을 이용하여 실험하였다.

$$\overline{w_i}\times(1-\frac{\beta}{2}) \le \overline{w_i} \le \overline{w_i}\times(1+\frac{\beta}{2}) \quad (9)$$



(그림 5) (a) 가우스 소거 그래프, (b) 고속 푸리에 변환 그래프

• 표준 태스크 그래프

알고리즘의 공정한 성능을 측정하기 위해 기존의 문헌과 연구에서 사용된 DAG 그래프를 인용하였다. 표준 태스크 그래프 프로젝트(STDGP)는 스케줄링 알고리즘의 성능을 공정하게 측정하기 위한 표준 DAG를 제공한다[12].

• 실제 응용프로그램 그래프

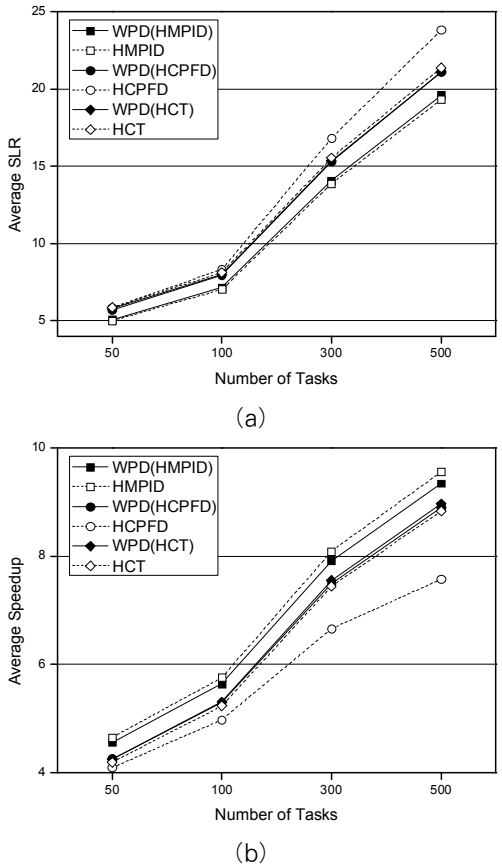
가우스 소거(Gaussian Elimination, GE) 알고리즘은 선형연립방정식의 해를 구하는데 사용하는 방법으로 전진소거와 후진대입을 이용하여 주어진 문제를 해결한다[13]. 가우스 소거를 사용하는 응용프로그램의 그래프에 포함되는 태스크의 수 v와 레벨의 수 l은 행렬의 크기 m에 따라 결정된다. $m = 5$인 경우의 그래프는 (그림 5) (a)와 같다. 그

래프에서 $T_{k,k}$는 피벗 칼럼 연산을 의미하고, $T_{k,j}$는 데이터 갱신을 의미한다.

(그림 5) (b)는 4개의 데이터 포인트를 갖는 그래프의 재귀적 1차원 고속 푸리에 변환(Fast Fourier Transform, FFT) 알고리즘을 그래프로 나타낸 것이다. 알고리즘은 재귀 호출과 버터플라이 연산의 두 단계로 구성된다[14]. 그림에서 점선을 기준으로 윗부분은 태스크를 재귀 호출하는 부분, 아랫부분은 버터플라이 연산을 하는 부분으로 나누어 볼 수 있다. 고속 푸리에 변환 그래프에서는 동일한 계층에 있는 모든 태스크의 계산비용이 동일하고 연속하는 두 계층을 잇는 에지의 통신비용 역시 동일하다.

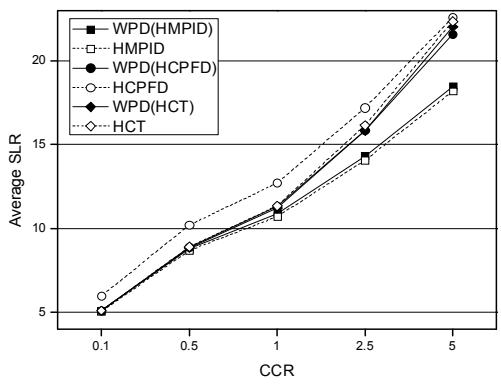
5.3 시뮬레이션 결과

(그림 6) (a)는 입력 DAG에 포함된 태스크 수의 변화에 따른 평균 SLR을 보여준다. 그림에서 괄호 안에 있는 알고리즘은 WPD 알고리즘과 결합한 프로세서 할당 단계를 제공하는 알고리즘을 의미한다. 예를 들어, WPD(HMPID)는 WPD 알고리즘과 HMPID 알고리즘의 프로세서 할당 단계를 결합한 것이다. 각 태스크 수에 대해 WPD(HMPID)={5.082, 7.147, 14.062, 19.589}, HMPID={5.000, 7.024, 13.861, 19.301}, WPD(HCPFD)={5.690, 7.931, 15.300, 21.078}, HCPFD={5.843, 8.326, 16.816, 23.805}, WPD(HCT) ={5.783, 7.990, 15.342, 21.112}, HCT={5.866, 8.112, 15.540, 21.363}이다. WPD 알고리즘을 태스크 복제만을 사용하는 HCPFD 알고리즘이나 HCT 알고리즘과 결합하였을 경우 향상된 성능을 보이는 것을 알 수 있다. 그러나 태스크의 복제와 삽입을 함께 사용하는 HMPID 알고리즘과 결합하였을 경우 상대적으로 성능이 하락하는 것을 확인할 수 있다. 이는 WPD 알고리즘이 우선순위 결정 단계에서 통신비용을 고려하지 않기 때문에 태스크의 삽입을 함께 고려하는 HMPID 알고리즘과 결합하였을 때 태스크의 삽입이 원활히 일어나지 못해 성능이 저하되는 것으로 예상할 수 있다.

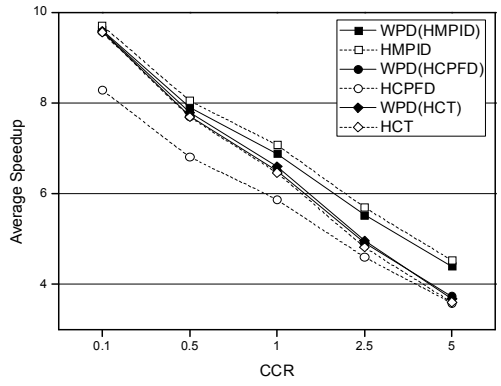


(그림 6) 태스크 수에 따른 (a) 평균 SLR, (b) 평균 Speedup

(그림 6) (b)는 입력 DAG에 포함된 태스크 수의 변화에 따른 평균 Speedup을 보여준다. 각 태스크 수에 대해 WPD(HMPID)={4.557, 5.638, 7.914, 9.343}, HMPID={4.649, 5.754, 8.089, 9.556}, WPD(HCPFD) ={4.254, 5.288, 7.498, 8.894}, HCPFD={4.089, 4.977, 6.657, 7.573}, WPD(HCT)={4.245, 5.305, 7.552, 8.966}, HCT={4.191, 5.231, 7.438, 8.832}이다. WPD 알고리즘을 HCPFD 알고리즘이나 HCT 알고리즘과 결합하였을 경우 향상된 성능을 보이지만, HMPID 알고리즘과 결합하였을 경우 상대적으로 성능이 저하되는 것을 확인할 수 있다.

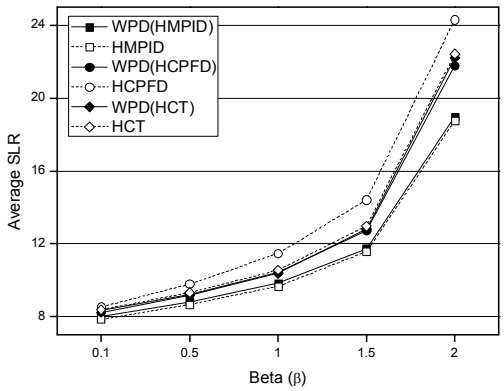


(a)

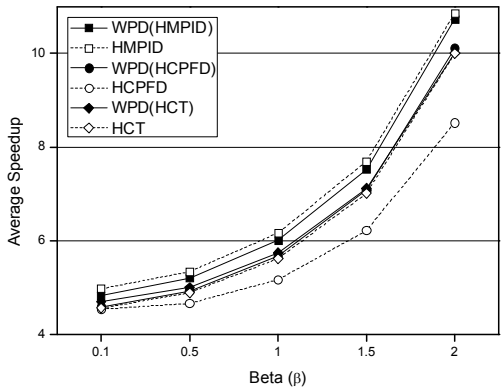


(b)

(그림 7) CCR에 따른 (a) 평균 SLR, (b) 평균 Speedup



(a)



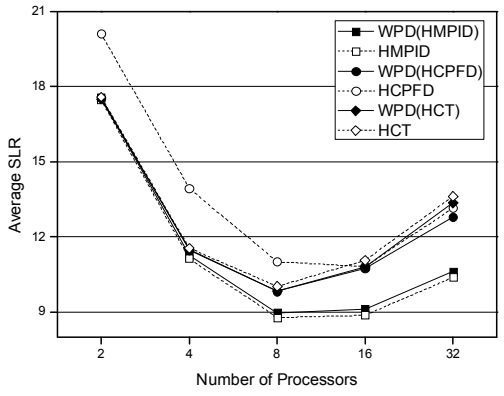
(b)

(그림 8) β에 따른 (a) 평균 SLR, (b) 평균 Speedup

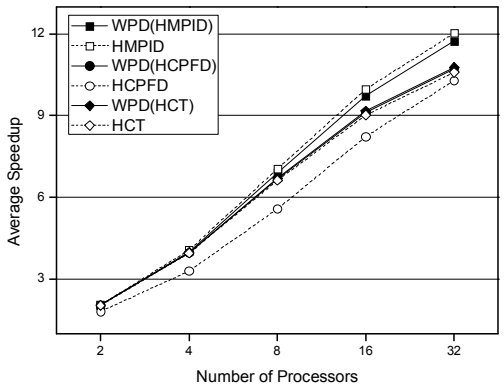
(그림 7) (a)는 CCR의 변화에 따른 평균 SLR의 변화를 보여준다. HMPID 알고리즘의 경우 WPD 알고리즘과 결합하였을 때 평균 1.51%의 성능 하락이 있었으며, HCPFD 알고리즘의 경우 평균 9.46%의 성능 향상이, HCT 알고리즘의 경우 평균 1.32%의 성능 향상이 있는 것을 확인할 수 있다. WPD 알고리즘을 HCPFD 알고리즘이나 HCT 알고리즘과 결합하였을 경우 향상된 성능을 보이지만, 삽입을 사용하는 HMPID 알고리즘과 결합하였을 경우 상대적으로 성능이 저하되는 것을 확인할 수 있다.

(그림 7) (b)는 CCR의 변화에 따른 평균 Speedup의 변화를 보여준다. HMPID 알고리즘의 경우 WPD 알고리즘과 결합하였을 때 평균 2.13%의 성능 하락이 있었으며, HCPFD 알고리즘의 경우평균 11.24%의 성능 향상이, HCT 알고리즘의 경우 평균 1.47%의 성능 향상이 있는 것을 확인할 수 있다. 전체적으로 평균 SLR 변화와 같은 추세를 보이며 WPD 알고리즘을 태스크 복제만을 사용하는 HCPFD 알고리즘이나 HCT 알고리즘과 결합하였을 경우 향상된 성능을 보이지만, 삽입을 사용하는 HMPID 알고리즘과 결합하였을 경우 상대적으로 성능이 하락하는 것을 확인할 수 있다.

(그림 8) (a)는 β의 변화에 따른 평균 SLR의 변화를 보여준다. HMPID 알고리즘의 경우 평균 1.52%의 성능 하락이 있었으며, HCPFD 알고리즘의 경우 평균 9.55%의 성능 향상이, HCT 알고리즘의 경우 평균 1.35%의 성능 향상이 있는 것을 확인할 수 있다. HMPID 알고리즘과 HCPFD 알고리즘



(a)



(b)

(그림 9) 프로세서 개수에 따른 (a) 평균 SLR, (b) 평균 Speedup

의 경우 프로세서 이질성도 값이 증가함에 따라 WPD 알고리즘이 더 좋은 성능을 보여주었지만 HCT 알고리즘의 경우 프로세서 이질성도 값이 증가함에 따라 기존의 알고리즘과 동일한 성능을 보이는 것을 확인할 수 있다.

(그림 8) (b)는 β의 변화에 따른 평균 Speedup의 변화를 보여준다. HMPID 알고리즘의 경우 평균 2.10%의 성능 하락이 있었으며, HCPFD 알고리즘의 경우 평균 11.29%의 성능 향상이, HCT 알고리즘의 경우 평균 1.51%의 성능 향상이 있는 것을 확인할 수 있다. HCPFD 알고리즘의 경우 프로세서 이질성도 값이 증가함에 따라 WPD 알고리즘이 더 좋은 성능을 보여주었지만 HCT 알고리즘의 경우 프로세서 이질성도 값이 증가함에 따라 기존의 알고리즘과 동일한 성능을 보이는 것을 확인할 수 있다.

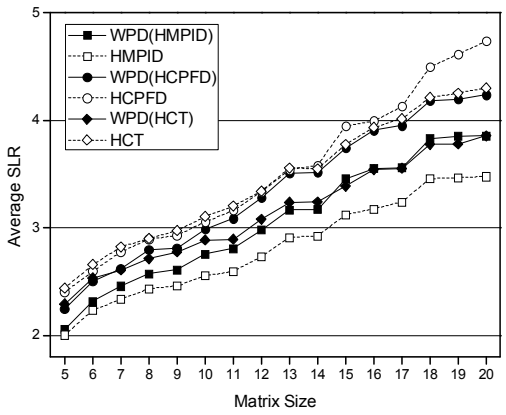
(그림 9) (a)는 프로세서 개수의 변화에 따른 평균 SLR의 변화를 보여준다. HMPID 알고리즘의 경우 평균 1.52%의 성능 하락이 있었으며, HCPFD 알고리즘의 경우 평균 10.69%의 성능 향상이, HCT 알고리즘의 경우 평균 1.33%의 성능 향상이 있는 것을 확인할 수 있다. 프로세서 개수가 16개를 넘어가는 경우 특정한 상황에서 매우 큰 스케줄 길이 비율이 생성되어 평균 스케줄 길이 비율이 다시 증가하는 모습을 보이고 있다.

(그림 9) (b)는 프로세서 개수의 변화에 따른 평균 Speedup의 변화를 보여준다. HMPID 알고리즘의 경우 평균 2.10%의 성능 하락이 있었으며, HCPFD 알고리즘의 경우 평균 11.34%의 성능 향상이, HCT 알고리즘의 경우 평균 1.41%의 성능 향상이 있는 것을 확인할 수 있다. 평균 Speedup의 경우 평균 SLR과 달리 프로세서 개수가 증가함에 따라 성능도 향상되는 것을 알 수 있다.

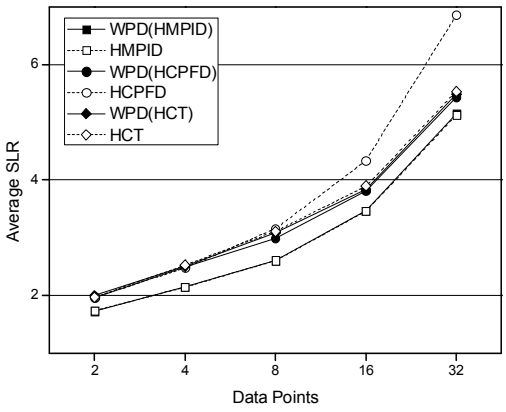
(그림 10) (a)는 가우스 소거법에 이용되는 행렬의 크기를 변화 시켰을 때 각 알고리즘의 평균 SLR을 보여준다. HMPID 알고리즘의 경우 평균 7.91%의 성능 감소가, HCPFD 알고리즘의 경우 평균 4.90%의 성능 향상이, HCT 알고리즘의 경우 평균 9.73%의 성능 향상이 있는 것을 확인할 수 있다. 특히 태스크 복제만을 사용하는 HCPFD 알고리즘과 HCT 알고리즘의 경우 행렬의 크기가 커질수록 WPD 알고리즘이 더 효율적인 것을 알 수 있다.

(그림 10) (b)는 고속 푸리에 변환 연산에 이용되는 데이터 포인트의 크기를 변화 시켰을 때 각 알고리즘의 평균 SLR을 보여준다. HMPID 알고리즘의 경우 평균 0.10%의 성능 감소가, HCPFD 알고리즘의 경우 평균 12.83%의 성능 향상이, HCT 알고리즘의 경우 평균 1.10%의 성능 향상이 있는 것을 확인할 수 있다.

(그림 11)은 WPD 알고리즘과 삽입 기반 프로세서 할당 정책을 결합했을 때, 입력 DAG의 태스크 수에 따른 평균 SLR을 보여준다. 그림에서 괄호 안의 Insertion은 WPD 알고리즘이 삽입 기반 프로세

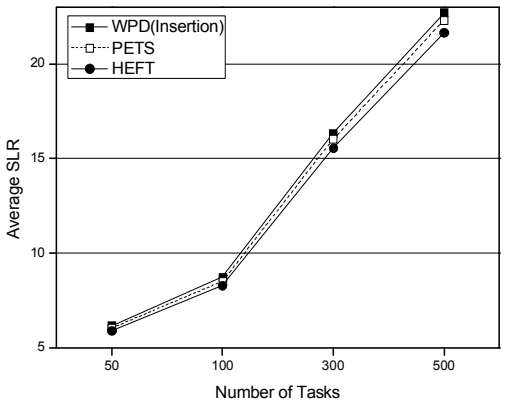


(a)



(b)

(그림 10) (a) GE의 행렬 크기, (b) FFT의 데이터 포인트에 따른 평균 SLR



(그림 11) 태스크의 삽입을 사용했을 경우 태스크 개수에 따른 평균 SLR

서 할당 정책과 함께 사용되었다는 것을 의미한다. HEFT 알고리즘과 PETS 알고리즘은 대표적인 리스트 스케줄링 알고리즘으로 삽입 기반 프로세서 할당 정책을 사용한다. 그림에서 볼 수 있듯이 WPD 알고리즘이 삽입 기반 프로세서 할당 정책과 함께 사용될 경우, 태스크 복제를 사용하는 프로세서 할당 정책과 함께 사용되었을 때와 다르게 기존의 리스트 스케줄링 알고리즘보다 성능이 저하되는 것을 확인할 수 있다. 이는 WPD 알고리즘이 우선순위 결정 단계에서 통신비용을 고려하지 않았기 때문에 태스크의 삽입 기회가 적어지면서 발생하는 성능 저하로 분석할 수 있다. WPD 알고리즘을 태스크의 삽입과 복제를 동시에 사용하는 HMPID 알고리즘과 결합하였을 때 성능이 저하되는 이유 역시 프로세서 할당 단계에서 태스크의 삽입이 원활히 일어나지 못해 발생하는 성능 저하로 해석할 수 있다.

6. 결 론

본 논문에서는 태스크 복제 기반 프로세서 할당 방법을 위한 태스크 우선순위 결정 알고리즘인 WPD 알고리즘을 제안하였다. WPD 알고리즘은 입력 DAG를 계층화 한 뒤 각 계층 내에서 태스크의 평균계산비용에 대한 내림차순으로 정렬하여 태스크의 우선순위를 결정한다. 기존 연구에서 제안한 태스크 복제 기반 스케줄링 알고리즘은 계산비용과 통신비용을 모두 고려하여 태스크의 우선순위를 결정하였지만 본 논문에서 제안하는 WPD 알고리즘은 계산비용을 이용하여 태스크의 우선순위를 결정하고, 계층화를 통해 서로 독립인 태스크들이 동시에 실행 가능하도록 하여 전체적인 성능을 향상시킨다.

제안하는 알고리즘의 성능 평가를 위해 표준 태스크 그래프에 다양한 매개변수를 적용하여 생성하고 실험하였다. 총 90,000개의 입력 DAG가 실험을 위해 사용되었으며, 의미 있는 성능 평가를 위해 각 매개변수의 변화에 따른 평균 SLR, Speedup의 변화를 비교하였다. 입력 DAG에 포함된 태스크

의 수에 따른 평균 SLR과 Speedup을 비교한 결과 WPD 알고리즘을 프로세서 할당 단계에서 태스크 복제만을 사용하는 HCPFD 알고리즘이나 HCT 알고리즘과 결합하는 경우 평균 9.58%, 1.31%의 성능 향상이 있었다. 하지만 WPD 알고리즘을 HEFT, PETS와 같이 삽입 기반 프로세서 할당 정책을 사용하는, 혹은 HMPID와 같이 태스크 삽입과 복제를 함께 사용하는 프로세서 할당 정책을 사용하는 알고리즘과 결합했을 경우에는 태스크의 삽입이 원활히 일어나지 못해 성능이 하락하는 것을 알 수 있었다.

본 논문에서 제안하는 WPD 알고리즘을 이용할 경우 전체 알고리즘을 새로 설계하지 않고 기존 태스크 복제 기반 프로세서 할당 방법에 WPD 알고리즘을 결합하는 방법만으로 손쉽게 태스크 복제 기반 프로세서 할당 방법을 사용하는 태스크 스케줄링 알고리즘의 성능을 향상시킬 수 있다. 정적 태스크 스케줄링 알고리즘의 경우 태스크 우선순위 결정 단계와 프로세서 할당 단계가 독립적으로 이루어져 있기 때문에 복잡한 과정을 거치지 않고 기존 태스크 스케줄링 알고리즘의 태스크 우선순위 결정 단계를 WPD 알고리즘으로 교체할 수 있다. 나아가 그리드 시스템, 클라우드 시스템 등 분산 이기종 컴퓨팅 시스템에 적용되어있는 태스크 복제 기반 태스크 스케줄링 알고리즘의 태스크 우선순위 결정 단계만을 변경하는 것으로 전체 태스크 스케줄링 알고리즘을 변경해야 하는 복잡한 과정 없이 분산 이기종 컴퓨팅 시스템의 성능을 끌어올릴 수 있을 것이라 기대된다.

참 고 문 헌

[1] J. G. Webster, Heterogeneous distributed computing, Encyclopedia of Electrical and Electronics Engineering, vol. 8, pp. 679-690, 1999.

[2] O. Sinnen, Task Scheduling For Parallel Systems, Wiley, 2007.

[3] J. D. Ullman, ‘NP-Complete Scheduling Problems.’, J. Computer and Systems Sciences, vol. 10, pp. 384-393, 1975.

[4] H. Topcuoglu, S. Hariri, and M. Y. Wu, ‘Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing.’, IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260-274, Feb. 2002.

[5] E. Ilavarasan, P. Thambidurai, and R. Mahilmannan, ‘Performance Effective Task Scheduling Algorithm for Heterogeneous Computing System.’, Parallel and Distributed Computing, International Symposium on Parallel and Distributed Computing, pp. 28-38, 2005.

[6] W. O. Yoon, J. H. Yoon, C. H. Lee, H. G. Gim, and S. B. Choi, ‘An Efficient List Scheduling Algorithm in Distributed Heterogeneous Computing System.’, Journal of IEEK : CI, vol.46, no.3, pp. 86-95, May. 2009.

[7] W. O. Yoon, I. S. Song, C. H. Lee, S. B. Choi, ‘An Efficient Task Scheduling Algorithm that Considers Insertion and Duplication in Heterogeneous Multi-Processor Systems’, Journal of KIISE : Computer Systems and Theory, vol. 38, pp. 67-79, 2011

[8] T. Hagras and J. Janecek, ‘A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems.’, Parallel and Computing, vol. 31, pp. 653-670, 2005.

[9] C. H. Liu, C. F. Li, K. C. Lai, and C. C. Wu, ‘A Dynamic Critical Path Duplication Task Scheduling Algorithm for Distributed Heterogeneous Computing Systems.’, International Conference on Parallel And Distributed Systems, vol. 1, pp. 365-374, 2006.

[10] L. Zhou and S. Shixin, 'Scheduling algorithm based on critical tasks in heterogeneous environments.', Journal of Systems Engineering and Electronics, vol. 19, no. 2, pp. 398-404, 2008.
[11] X. Tang, K. Li, G. Liao, R. Li, 'List scheduling with duplication for heterogeneous computing systems', Journal of Parallel Distributed Computing, vol. 70, pp. 323-329, 2010.
[12] http://www.kasahara.elec.waseda.ac.jp.
[13] M. Cosnard, M. Marrakchi, Y. Robert, and D. Trystram, 'Parallel Gaussian Elimination on an MIMD Computer.', Parallel Computing, vol. 6, pp. 275-295, 1988.
[14] Y. Chung and S. Ranka, 'Applications and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors.', Proc. Supercomputing, pp. 512-521, Nov. 1992.

◐ 저 자 소 개 ◑

송 인 성
2009년 인하대학교 전자공학과 졸업(학사)
2011년 인하대학교 대학원 전자공학과 졸업(석사)
2011년~현재 인하대학교 대학원 전자공학과 박사과정
관심분야 : 병렬 및 분산 처리 시스템, 컴퓨터 아키텍처
E-mail : nicvirus@inha.edu

윤 완 오
2000년 경기대학교 전자공학과 졸업(학사)
2002년 인하대학교 대학원 전자공학과 졸업(석사)
2010년 인하대학교 대학원 전자공학과 졸업(박사)
2010년~현재 인하대학교 정보전자공동연구소 연구교수
관심분야 : 병렬 및 분산 처리 시스템, 컴퓨터 아키텍처
E-mail : wanoh38@paran.com

이 창 호
2008년 청주대학교 전자공학과 졸업(학사)
2010년 인하대학교 대학원 전자공학과 졸업(석사)
2010년~현재 인하대학교 대학원 전자공학과 박사과정
관심분야 : 컴퓨터 아키텍처, 컴퓨터 네트워크, 병렬 및 분산 처리 시스템
E-mail : lchh0902@nate.com

◐ 저 자 소 개 ◑

최 상 방

1981년 한양대학교 전자공학과 졸업(학사)
1981~1986년 LG 정보통신(주)
1988년 University of Washington 졸업(석사)
1990년 University of Washington 졸업(박사)
1991년~현재 인하대학교 전자공학과 교수
관심분야 : 컴퓨터 아키텍처, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, Fault-tolerant Computing
E-mail : sangbang@inha.ac.kr