논문 2009-46CI-3-11

분산 이기종 컴퓨팅 시스템에서 효율적인 리스트 스케줄링 알고리즘

(An Efficient List Scheduling Algorithm in Distributed Heterogeneous Computing System)

윤 완 오*, 윤 정 희*, 이 창 호*, 김 효 기*, 최 상 방**

(Wan-Oh Yoon, Jung-Hee Yoon, Chang-Ho Lee, Hyo-Gi Gim, and Sang-Bang Choi)

요 약

이기종 컴퓨팅 환경에서 방향성 비순환 그래프(directed acyclic graph DAG)의 효율적인 스케줄링은 시스템의 성능을 높게 만드는데 매우 중요한 역할을 한다. 이기종의 컴퓨팅 환경에서 DAG로 표현되는 프로그램의 최적 스케줄링 방법을 찾는 것은 잘 알려진 '정해진 시간 내에 해결하기 어려운 문제(NP-complete)' 이다. 본 논문은 분산 이기종 컴퓨팅 시스템에서 병렬로 실행 가능한 프로그램을 위한 새로운 리스트 스케줄링 알고리즘인 HRPS(Heterogeneous Rank-Path Scheduling)를 제안하였다. HRPS의 가장 궁극적인 목적은 프로그램의 실행시간을 최소화하는 것이다. 알고리즘의 성능을 위해 DAG 입력 그래프를 이용하여 기존에 제안되어진 CPOP, HCPT, FLB 알고리즘과 스케줄의 길이를 비교한 결과 성능 향상의 결과를 얻을 수 있었다.

Abstract

Efficient DAG scheduling is critical for achieving high performance in heterogeneous computing environments. Finding an optimal solution to the problem of scheduling an application modeled by a directed acyclic graph(DAG) onto a set of heterogeneous machines is known to be an NP-complete problem. In this paper we propose a new list scheduling algorithm, called the Heterogeneous Rank-Path Scheduling(HRPS) algorithm, to exploit all of a program's available parallelism in distributed heterogeneous computing system. The primary goal of HRPS is to minimize the schedule length of applications. The performance of the algorithm has been observed by its application to some practical DAGs, and by comparing it with other existing scheduling algorithm such as CPOP, HCPT and FLB in term of the schedule length. The comparison studies show that HRPS significantly outperform CPOP, HCPT and FLB in schedule length.

Keywords: list scheduling, heterogeneous, DAG scheduling, parallel processing, speedup

I. 서 론

일반적으로 분산 이기종 컴퓨팅 시스템(distributed heterogeneous computing system DHCS)은 다른 처리 능력을 가지는 여러 프로세서들이 속도가 다른 네트워크로 연결되어 구성된다. 디지털 영상처리, 컴퓨터 비전, 기상 모델, 유체 흐름(fluid flow), 이미지 처리, 멀티미디어 정보처리 및 통신 분야 등에서 대규모의 데이터베이스와 그 처리 알고리즘의 복잡성 때문에 병렬 고속 연산이 절실히 요구되고 있으며 이를 위해 병렬 처리 및 분산 이기종 컴퓨팅 시스템이 주목받고 있다. 분산 이기종 컴퓨팅에서 응용 프로그램을 실행하기 위해서는 응용 프로그램을 병렬 프로그램화하여야 한다. 이와 같은 병렬 프로그램은 응용 프로그램의 태스크를 표현하는 노드(node), 다른 태스크와의 데이터 의존도를 표현하는 에지(edge)로 구성되는 DAG로 도식화하여 표현할 수 있다. 분산 이기종 컴퓨팅 시스템에서 DAG 실행

* 학생회원, ** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering Inha University)
※ 이 논문은 인하대학교의 지원에 의하여 연구되었습니다.
접수일자 : 2009년3월2일, 수정완료일: 2009년4월28일

에 대한 성능은 노드를 각 프로세서에 할당하는 스케줄링 알고리즘 방법에 많이 의존된다. 스케줄링의 목적은 상호 연관되어진 노드의 요구 조건을 만족하면서 각 프로세서에 노드를 적절히 할당하여 프로그램의 전체 실행 시간(makespan)을 최소화하는 것이다. 최적의 스케줄링은 분산 이기종 컴퓨팅 시스템의 성능 향상을 가져다주지만 다항 시간(polynomial time)내에 결과를 얻을 수 없는 NP-complete이다[1~3]. 그래서 많은 연구들이 만족할만한 시간-복잡도와 최적에 가까운 해결책을 얻기 위해 NP-complete 문제를 해결하는데 중점을 두고 있다. 최근 최적의 스케줄링을 위해 활발히 연구되고 있는 분야는 스케줄링을 실행하는 시점에 따라 정적, 동적으로 나눌 수 있으며 정적 알고리즘은 다시 리스트 스케줄링, 클러스터 스케줄링, 노드 복제 스케줄링으로 나눌 수 있다. 리스트 스케줄링은 노드 간의 상호 제약을 충족시키면서 몇몇 우선순위 함수에 기초하여 노드의 우선순위를 결정한다. 리스트 스케줄링에서 사용되는 우선순위 함수는 태스크의 계산비용과 통신비용에 기초를 두며 최우선 노드부터 적절한 프로세서를 선택하여 할당하게 된다. 리스트 스케줄링은 비교적 다른 알고리즘에 비해 낮은 복잡도를 갖는다[4~8]. 클러스터 스케줄링은 태스크들 사이의 통신비용을 고려하여 여러 개의 태스크를 하나의 클러스터로 묶어 최적의 프로세서에 할당함으로써 통신시간을 줄여 최적의 결과를 얻기 위한 알고리즘이다[9~11]. 노드 복제 스케줄링은 노드들 간의 상호 의존성에 의한 통신시간을 감소시키기 위해 선행 노드의 복제를 통해 스케줄링을 수행하는 알고리즘이며 다른 스케줄링에 비해 복잡도가 커지는 단점은 있으나 스케줄링의 결과는 다른 알고리즘에 비해 향상된 결과를 보인다[12~14].

본 논문에서는 리스트 스케줄링의 한 방법으로써 새로운 정적 스케줄링 알고리즘인 HRPS (Heterogeneous Rank-Path Scheduling)을 제안하였다. 일반적으로 주어진 입력 그래프는 시작노드부터 마지막 노드까지 여러 개의 경로가 존재할 수 있으며 그 중에서 임계 경로(critical path)가 전체 실행 시간의 길이를 결정하므로 임계 경로에 대한 고려가 중요한 역할을 한다. 기존의 알고리즘들은 임계경로만을 고려하여 임계경로를 이루는 노드들에 먼저 우선순위를 부여하여 스케줄링 하였으나 HRPS 알고리즘은 입력 그래프의 임계경로(critical path) 뿐만 아니라 시작노드부터 마지막 노드까지의 모든 경로에 순위를 정해 노드의 우선순위를 결정하는 방법을 사용한다.

제안된 알고리즘의 성능을 확인하기 위해 여러 가지 매개변수에 의해 만들어진 임의의 그래프와 벤치마크로부터 추출된 태스크 그래프를 활용하여 기존의 대표적인 리스트 스케줄링 알고리즘인 CPOP[6], HCPT[7] 그리고 FLB[8]와 성능 비교를 하였다. 비교 결과 본 논문에서 제안한 HRPS 알고리즘의 성능이 다른 알고리즘의 성능에 비해 향상된 것을 확인할 수 있었다.

본 논문은 다음과 같이 이루어져 있다. Ⅱ장에서는 본 논문에서 사용될 문제와 용어를 설명하고 Ⅲ장에서는 기존의 대표적인 리스트 스케줄링 알고리즘인 CPOP, HCPT을 소개한다. Ⅳ장에서는 본 논문에서 제안한 알고리즘을 설명하고 Ⅴ장에서는 실제 사용되는 어플리케이션의 태스크 그래프와 임의의 태스크 그래프를 기반으로 시뮬레이션을 통해 제안된 알고리즘의 성능 분석을 기술한다. 마지막으로 Ⅵ장에서 결론을 맺는다.

Ⅱ. 문제 정의

이번 장에서는 본 논문에서 제안한 스케줄링을 수행하기 위해서 필요한 시스템 환경, 입력 데이터, 매개변수를 정의한다. 분산 이기종 컴퓨팅 시스템 환경은 k개의 이기종 프로세서 p들이 완전연결(fully connected)로 구성된 집합 P로 나타낼 수 있으며 프로세서간의 통신 채널 및 대역은 충분하다고 가정한다. 또한 각 프로세서는 태스크의 수행과 통신이 동시에 이루어지며 태스크가 수행하는 동안 어떠한 방해없이 수행이 가능하고 수행결과는 즉시 다른 프로세서에 보낼 수 있다고 가정한다. 분산 이기종 컴퓨팅 시스템에서 실행할 수 있는 프로그램은 그림 1과 같이 방향성 비순환 그래프(DAG) $G=(V,E)$로 표현할 수 있다. 여기서 V는 노드 혹은 태스크(본 논문에서는 노드와 태스크를 같은 뜻으로 간주한다) v의 집합이고 노드들 중에서 i 번째 노드는 v_i로 표현한다. E는 통신 에지(communication edge) e의 집합이며 노드 v_i에서 v_j로의 방향성을 갖는 에지 $e_{i,j}$가 존재한다면 v_i는 v_j의 부모노드라고 부르고, v_j는 v_i의 자식노드라고 부른다. 그림 1의 노드 1처럼 부모노드가 존재하지 않는 노드를 시작노드(entry node) v_s라 하고 노드 10처럼 자식노드가 없는 노드를 출력노드(exit node) v_e라 정의한다. 본 논문에서 제안한 스케줄링을 하기 위해서는 오직 한 개의 입력노드와 출력노드가 필

요하다. 만약 2개 이상의 입력노드와 출력노드가 있다면 이 노드들을 묶을 수 있는 의사노드(pseudo node)를 추가할 수 있으며 의사노드의 계산비용과 통신비용은 모두 0값을 가지고 이 노드는 스케줄링 상에 아무런 영향을 주지 않는다고 가정한다.

W는 표 1과 같이 노드 $v_i \in V$가 프로세서 $p_m \in P$에서 실행될 때의 계산비용(computation cost) $w_{i,m}$로 이루어진 $|V| \times |P|$ 크기의 계산비용 행렬이다. 스케줄링 알고리즘을 수행하기 전에 각 노드의 평균 계산비용을 계산해야 하는데 예를 들어 k개의 프로세서에서 노드 v_i 의 평균 계산비용 $\overline{w_i}$는 식(1)과 같다.

$$\overline{w_i} = \sum_{m=1}^{k} \frac{w_{i,m}}{k} \tag{1}$$

두 개의 프로세서 사이에 전송되는 데이터의 바이트당 전송비용은 $k \times k$ 크기를 가지는 행렬 R로 표현되고 각 프로세서의 통신 시작비용(startup cost)은 k차원 벡터 S로 주어진다. 이 때, 프로세서 p_m에 스케줄링 된 노드 v_i로부터 프로세서 p_n에 스케줄링된 노드 v_j까지 μ바이트 크기에 데이터를 전송할 때 그에 해당하는 에지 $e_{i,j} \in E$의 통신비용 $c_{(i,m)(j,n)}$는 식 (2)와 같이 정의할 수 있다.

$$c_{(i,m)(j,n)} = \begin{cases} 0 & \text{if } m = n \\ S_m + R_{m,n} \cdot \mu_{i,j} & otherwise \end{cases} \tag{2}$$

여기서 S_m은 프로세서 p_m의 통신 시작시간(sec), $\mu_{i,j}$는 노드 v_i부터 노드 v_j로 전달되는 데이터 전송 양

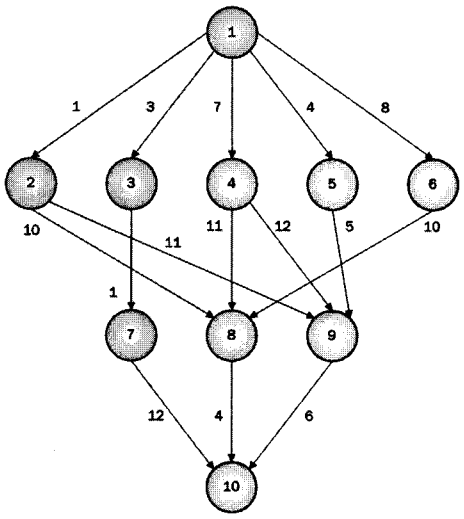


그림 1. 10개의 노드를 가지는 DAG
Fig. 1. A example of DAG with 10 nodes.

표 1. 계산 비용 행렬(W)과 평균 계산비용($\overline{w_i}$)
Table 1. Computation cost matrix(W) and average computation cost($\overline{w_i}$).

node	p_1	p_2	p_3	$\overline{w_i}$
v_1	22	24	13	19.667
v_2	5	5	17	9.0
v_3	13	21	14	16.0
v_4	21	22	10	17.667
v_5	17	20	5	14.0
v_6	23	21	6	16.667
v_7	5	4	22	10.333
v_8	15	13	24	17.333
v_9	14	13	20	15.667
v_{10}	16	16	19	17.0

(bytes) 그리고 $R_{m,n}$은 p_m으로부터 p_n까지의 바이트 당 통신비용(sec/byte)이다. 프로세서 p_m과 p_n이 같은 프로세서일 경우 통신비용은 0값을 가지며 서로 다른 프로세서라면 통신비용은 통신 시작시간과 μ 바이트 크기의 데이터가 전달되는데 필요한 시간의 합으로 나타낼 수 있다. 그림 1의 입력 그래프 DAG에서 에지 $e_{i,j} \in E$에 표현된 수치는 평균 통신비용 $\overline{c_{i,j}}$을 식 (3)을 이용하여 구해진 값을 표현한 것이다.

여기서 $\overline{S}$는 모든 프로세서의 평균 통신 시작시간이며 $\overline{R}$은 모든 프로세서에 대한 바이트 당 평균 통신비용이다.

$$\overline{c_{i,j}} = \overline{S} + \overline{R} \cdot \mu_{i,j} \; where, \begin{cases} \overline{S} = \sum_{m=1}^{k} \frac{S_m}{k} \\ \overline{R} = \sum_{m=1}^{k} \sum_{n=1}^{k} \frac{R_{m,n}}{(k(k-1))} \end{cases} \tag{3}$$

III. 관련 연구

지금까지 연구된 리스트 스케줄링 알고리즘은 여러 가지가 있으나 본 논문에서 제안한 HRPS 스케줄링 알고리즘과 성능 비교를 하기 위해 사용된 리스트 스케줄링 알고리즘인 CPOP(Critical path on a processors), HCPT(Heterogeneous Critical Parent Trees), FLB(Fast load Balancing)를 간단하게 설명한다.

CPOP(Critical path on a processors) : CPOP 스케줄링 알고리즘도 일반적인 리스트 스케줄링과 마찬가지로

두 단계로 이루어져 있다. 첫 번째 단계인 노드 우선순위 단계에서는 시작 노드부터 시작해서 재귀적으로 통신비용과 계산비용의 합을 이용하여 $rank_d$ 값을 계산하고 마찬가지로 마지막 노드부터 시작노드까지 통신비용과 계산비용을 이용하여 $rank_u$값을 계산한다. $rank_u$와 $rank_d$의 합을 이용하여 각 노드의 우선순위를 결정한다. 두 번째 단계는 프로세서 할당 단계이다. 우선 순위가 가장 높은 노드를 선택하고 선택된 노드가 임계경로에 있는 노드라면 한 개의 최적 프로세서에 할당한다. 즉 임계경로의 노드는 한 개의 최적 프로세서에 할당하는 방식이다. 임계경로의 노드를 가급적 빨리 끝내는 것이 전체 스케줄 길이를 단축하는데 유리하다고 보고 임계경로 노드의 수행시간 합이 가장 작은 최적의 프로세서에 할당하는 것이다. 임계경로에 위치하는 노드가 아니라면 삽입정책(insertion-based policy)을 이용하여 프로세서에 할당한다. 알고리즘의 시간-복잡도는 $O(v^2 \times p)$의 크기를 갖는다.

HCPT(Heterogeneous Critical Parent Trees) : HCPT 알고리즘은 노드 우선순위 단계에서 가장 빠른 평균 시작시간(AEST)과 가장 늦은 평균 시작시간(ALST)의 차가 0인 노드를 임계 경로의 노드로 결정한다. 임계경로를 이루는 노드들을 스택에 넣고 비어있는 큐를 만든다. 스택의 최상위에 있는 노드의 부모노드가 큐에 있는지 확인한다. 큐에 부모노드가 없다면 부모노드를 스택에 $push$한다. 큐에 부모노드가 있다면 스택의 최상위 노드를 pop하여 큐에 넣고 스택이 빌 때까지 이와 같은 동작을 반복한 후 큐에 들어있는 순서대로 노드의 우선순위가 결정된다. 두 번째 단계에서는 큐에 있는 노드를 하나씩 가장 빠른 시간에 끝낼 수 있는 프로세서에 할당하게 된다. 이 알고리즘의 시간-복잡도는 $O(v^2 \times p)$이다. 이 알고리즘의 문제점은 부모노드가 많을 경우 어떤 부모노드를 먼저 선택할 지에 대한 정의가 없다. 임의대로 부모노드를 선택할 경우 노드의 우선순위는 변화할 수 있으며 그로 인한 성능향상을 기대할 수 없다.

FLB(Fast load Balancing) : FLB 알고리즘은 각 단계마다 스케줄되기 위한 준비-노드(ready-node)들이 포함된 준비-리스트(ready-list)를 이용한다. 여기서 준비-노드는 노드의 모든 부모노드가 스케줄된 노드를 의미한다. 각 단계마다 모든 프로세서에 대해 준비-노드의 실행 완료 시간을 계산하고 최소의 실행 완료 시간이 되는 프로세서에 준비-노드를 할당하는 알고리즘이다.

Ⅳ. 제안된 스케줄링 알고리즘

본 논문에서 제안한 HRPS 알고리즘은 리스트 스케줄링 알고리즘이다. 따라서 노드의 우선순위를 정하기 위한 우선순위 결정 단계와 결정된 우선순위로 노드를 프로세서에 할당하는 프로세서 선택 단계로 이루어져 있다. 우선순위 결정 단계에서는 DAG의 시작 노드부터 마지막 노드까지의 모든 경로를 구하고 구해진 경로의 순위에 따라 노드의 우선순위를 결정한다. 프로세서 할당 단계에서는 기존의 연구에서 사용되어진 삽입기반정책을 사용하여 프로세서에 태스크를 할당한다.

4.1 노드 우선순위 결정 단계

본 논문에서는 입력 그래프 DAG을 이루는 각 노드의 우선순위를 결정하기 위해 통신비용과 계산비용을 이용해 각 노드의 우선순위 값을 계산한다. 우선순위 값은 출력노드부터 시작해서 계산되며 식 (4)와 같이 정의된 우선순위 함수 $UpRank(v_i)$을 사용한다.

$$UpRank(v_i) = \overline{w_i} + \max_{v_j \in succ(v_i)} \{\overline{c_{i,j}} + UpRank(v_j)\} \quad where,\ v_i \neq v_s \tag{4}$$

여기서 $succ(v_i)$는 노드 v_i의 자식노드의 집합이며 $\overline{c_{i,j}}$는 에지 $e_{i,j}$의 통신비용의 평균이고 $\overline{w_i}$는 노드 v_i의 계산비용의 평균이다. 출력노드부터 재귀적으로 계산하기 때문에 출력노드의 $UpRank(v_e) = \overline{w_e}$ 으로 정의되고 시작노드 v_s 을 제외한 모든 노드의 우선순위 값을 계산한다. 주어진 DAG의 시작노드 v_s 을 제외한 모든 노드의 우선순위 값을 $UpRank(v_i)$을 이용하여 재귀적으로 계산한 후 시작노드의 우선순위 값 $Rank_l(v_s)$값을 식 (5)을 이용하여 계산한다.

$$Rank_l(v_s) = c_{v_s, succ(v_s)} + UpRank(succ(v_s)) \quad where,\ l = 1 \ldots |succ(v_s)| \tag{5}$$

$Rank_l(v_s)$는 v_s의 자식노드의 개수($|succ(v_s)|$)와 같은 수로 만들어 진다. $Rank_l(v_s)$에 의해 결정되는 경로를 순위경로($Rank\ Path$) RP_l로 정의한다. 즉, RP_l은 주

표 2. $UpRank(v_i)$, $Rank_l(v_s)$, RP_l의 결과 값
Table 2. Result value of $UpRank(v_i)$, $Rank_l(v_s)$, RP_l.

node	$UpRank(v_i)$
2	58.667
3	56.333
4	68.334
5	57.667
6	65.11
7	39.333
8	38.333
9	38.667
10	17.0

$Rank_l(v_s)$	value	RP_i	path
$Rank_1(v_s)$	95.001	RP_1	$v_s - v_4 - v_9 - v_e$
$Rank_2(v_s)$	92.777	RP_2	$v_s - v_6 - v_8 - v_e$
$Rank_3(v_s)$	81.334	RP_3	$v_s - v_5 - v_9 - v_e$
$Rank_4(v_s)$	79.334	RP_4	$v_s - v_2 - v_9 - v_e$
$Rank_5(v_s)$	79	RP_5	$v_s - v_3 - v_7 - v_e$

어진 DAG의 시작노드부터 출력노드까지의 모든 경로를의미한다. $Rank_1(v_s) \geq Rank_2(v_s) \ldots \geq Rank_{|succ(v_s)|}(v_s)$을 만족할 경우 이에 대응하는 순위경로도 $RP_1 \geq RP_2 \ldots \geq RP_{|succ(v_s)|}$을 만족하며 RP_1는 주어진 DAG의 임계경로 $CP(Critical\ Path)$이다. 표 2는 그림 1에서 주어진 DAG 그래프와 표 1의 실행시간을 이용하여 $UpRank(v_i)$, $Rank_l(v_s)$, RP_l을 계산한 결과이다.

본 논문에서 주어진 그림 1의 DAG 그래프에서 모든 노드들은 순위경로 RP_l에 포함되어 있다. 하지만 순위경로에 포함이 되지 않는 노드들이 존재하는 DAG 그래프도 있을 수 있다. 따라서 순위경로 RP_l 에 속하지 않는 노드들을 $ORPN(Out\ Rank\ Path\ Nodes)$으로 정의하고 $ORPN$ 노드들을 $UpRank(v_i)$의 값을 이용하여 내림차순으로 정렬한다. 내림차순으로 정렬된 $ORPN$ 노드의 집합을 RP_{l+1}로 정의한다. 따라서 입력 그래프의 순위경로의 수는 입력노드 v_s의 자식노드의 수 $|succ(v_s)|$ + 1이 된다. 스택을 순위경로의 수만큼 생성하고 생성된 스택과 스택의 집합을 $s_{l+1} \in S$로 정의 한다. 그 후에 순위경로의 마지막 노드들부터 각각의 스택에 $push$한다. 즉 스택 $s_1, s_2, s_3, s_4, s_5, s_6$에 순위경로 $RP_1, RP_2, RP_3, RP_4, RP_5, RP_6$의 노드들이 입력된다. 표 2에서 주어진 예제에서 순위경로 $RP_6 = \{\ \}$의 값을 갖는다.

비어있는 큐(queue) L을 만들고 s_1 스택부터 $top(s_1)$을 이용하여 스택의 최상위에 있는 노드의 부모노드가 L에 있는지 없는지 확인한다. 부모노드가 모두 L에 있을 경우에는 $pop(s_1)$을 이용하여 노드를 s_1으로부터 pop하고 L에 삽입한다. 만일 삽입하려는 노드의 부모노드가 L에 없을 경우에는 다음 스택을 선택한다. $top(s_1)$을 이용하여 부모노드가 L있는지 없는지 확인할 때 만일 이미 L에 자신의 노드가 존재한다면 pop을 이용해 스택에서 제거한다. 모든 스택의 노드가 비어질 때까지 반복하여 수행한다. 표 2에서 L에 삽입되는 순위경로의 순서는 $RP_1, RP_2, RP_3, RP_4, RP_5, RP_6$으로 반복되고 이에 대응하는 스택도 $s_1, s_2, s_3, s_4, s_5, s_6$ 순으로 선택된다. 스택 s_1의 v_s , v_4 노드를 L에 삽입하고 다음으로 v_9 노드를 L에 삽입할 때 L에 v_9의 부모노드들 중에서 v_2 ,v_5 노드가 없기 때문에 다음 스택인 s_2을 선택한다. s_2의 v_s노드는 이미 L에 있기 때문에 삽입하지 않고 다음 노드인 v_6을 삽입한다. 마찬가지로 s_2의 노드 v_8은 부모노드 중에서 v_2와 v_4 가 없기 때문에 s_3 스택을 선택하여 위의 과정을 스택이 모두 비어질 때까지 반복한다. 최종적으로 L은 $\{v_s, v_4, v_6, v_5, v_2, v_9, v_3, v_7, v_8, v_e\}$의 값을 가지게 된다.

4.2 프로세서 선택 단계

노드 우선순위 결정 단계에서 각 노드의 우선순위를 결정하고 나면, 우선순위가 가장 높은 노드부터 적절한 프로세서를 선택하여 할당하게 된다. 기존의 연구에서 프로세서를 선택하여 노드를 할당하기 위한 방법은 크게 삽입 기반 정책과 비 삽입 기반 정책으로 나눌 수 있다. 삽입 기반 정책은 하나의 프로세서에 이미 할당되어진 두 노드 사이에 여유공간이 있고 선행 제약을 만족한다면 노드를 삽입하는 정책이다. 일반적으로 삽입 기반 정책이 비 삽입 기반 정책에 비해 효율적인 결과를 보인다[15]. 본 논문에서는 노드의 실행 완료 시간에 최소인 프로세서에 기존에 연구되어진 삽입 기반 정책을 사용하여 노드를 프로세서에 할당한다. 식 (6), (7)은 노드의 초기 시작 시간 est(earliest start time)과 노드의 실행 완료 시간ect(earliest completion time)을 의미한다. 여기서 $pred(v_i)$는 노드 v_i의 부모노드의 집합을 의미하며 $PA[p_m]$은 프로세서 p_m에서 노드를 실행할 수 있는 실행 가능 시간을 의미한다. 처음 시작 노드가

$$est(v_i, p_m) = \begin{cases} 0 & \text{if } v_i = v_s \\ \max\limits_{v_j \in pred(v_i)} \{PA[p_m], \max(ect(v_j) + k \cdot c_{j,i})\} & otherwise \end{cases} \quad (6)$$

$$ect(v_i, p_m) = est(v_i, p_m) + w_{i,m} \quad (7)$$

프로세서 p_m에서 실행될 경우 $est(v_s, p_m) = 0$의 값을 가지게 된다. $\max(ect(v_j) + k \cdot c_{j,i})$은 v_i 의 부모 노드

Set the computation costs of node and communication cost of edges with mean values.
Compute $UpRank(v_i)$ and $Rank_l(v_s)$ for each node according to Eqns. (4) and (5).
determine $RP_{|succ(v_s)|+1}$ using $Rank_l(v_s)$ and $UpRank(v_i)$
build $s_{|succ(v_s)|+1}$ using $RP_{|succ(v_s)|+1}$
$L = \{\ \}$
$l = 1$
While S is not empty do
    if there is an unlisted parent of $top(s_l)$ then
        if $l = |succ(v_s)| + 1$ then
            $l = 1$
        else
            $l = l + 1$
    else
        $n = pop(s_l)$
        if $n \notin L$ then
            enqueue n to L
endwhile

while there are unscheduled node in the list L do
    Select the first node, v_i, from the list scheduling.
    for each processor p_m in the processor set do
        compute $est(v_i, p_m)$ value using the insertion-based scheduling policy.
    assign node v_i to the processor p_j that minimizes ect of node v_i.
endwhile

그림 2. HRPS 알고리즘의 의사코드
Fig. 2. pseudo code of the HRPS algorithm.

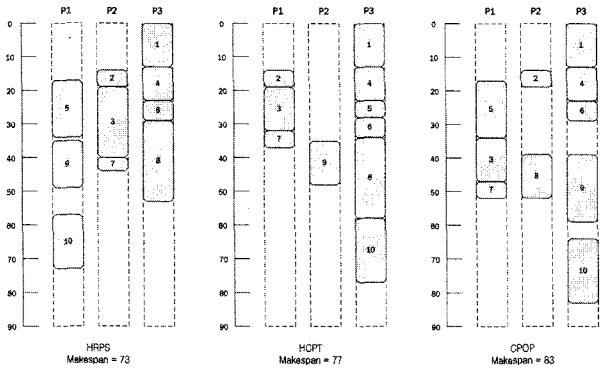


그림 3. HRPS, HCPT, CPOP 알고리즘의 스케줄 길이
Fig. 3. The schedule length generated by HRPS, HCPT, CPOP.

중에서 가장 늦게 데이터를 전달하는 노드를 의미하며 여기서 상수 k는 부모노드가 같은 프로세서에 할당되었을 경우 0의 값을 가지고 그 외에는 1의 값을 가지게 된다. 노드 실행 완료 시간 ect는 노드의 시작 시간에 실행시간을더한 값으로 표현된다. 따라서 노드의 프로세서 선택은 $ect(v_i, p_m)$ 가 최소가 되는 프로세서를 선택하여 노드를 할당하게 된다. 그림 2는 HRPS 알고리즘의 의사코드이다. 그림 3은 본 논문에서 제안한 HRPS 알고리즘과 HCPT, CPOP의 스케줄 길이를 그림 1의 DAG 그래프와 표 1의 계산비용을 이용하여 계산한 결과를 보여준다. 그림에서 HRPS는 73의 결과를 얻어 77의 HCPT와 83의 CPOP보다 성능 향상이 있는 것을 확인할 수 있다.

V. 성능 분석 및 평가

이번 장에서는 벤치마크 프로그램들로부터 생성된 DAG을 이용하여 제안한 알고리즘과 기존의 알고리즘인 HCPT, CPOP, FLB와 성능 비교 결과를 기술한다.

5.1 비교 기준

알고리즘들의 성능 비교를 위한 척도들이 기존에 제시된 바 있다[4~9]. 본 논문에서는 다음의 기준들에 의해 각 알고리즘의 성능을 비교하였다.

· Makespan

makespan은 주어진 입력 그래프 DAG의 수행이 완료된 시간을 의미하며 다음과 같이 마지막 노드 v_e가

수행 완료된 시간으로 표현될 수 있다.

$$makespan = ect(v_e, p_m) \tag{8}$$

· Schedule Length Ratio (SLR)

알고리즘의 성능 측정은 makespan을 주된 방법으로 이용한다. 하지만 각각 다른 속성을 가진 많은 양의 DAG 그래프들이 이용되기 때문에, Schedule Length Ratio(SLR)라 불리는 하한(lower bound) 값으로 makespan을 표준화시켜야 한다. 알고리즘의 SLR값은 다음 식으로 정의된다.

$$SLR = \frac{makespan}{\sum_{v_i \in CP} \min_{p_m \in P}\{w_{i,m}\}} \tag{9}$$

분모는 주어진 DAG 그래프에서 임계경로를 이루는 노드들의 최소 계산비용의 합이다. 하한 값을 분모로 이용하기 때문에 어떤 알고리즘을 이용하더라도 알고리즘의 SLR은 1보다 작아질 수 없다. 알고리즘들 중에서 가장 낮은 SLR을 만들어 내는 스케줄링 알고리즘이 가장 좋은 성능의 알고리즘이 된다.

· Average Speedup

speedup은 순차 실행 시간(sequential execution time)을 병렬 실행 시간(parallel execution time)으로 나눈 값으로 정의된다. 순차 실행 시간은 모든 노드들을 계산비용의 합이 최소가 되는 단일 프로세서에 할당하는 방법으로 계산하고 병렬 실행 시간은 makespan을 의미한다. 따라서 Speedup이 클수록 성능이 더 좋은 알고리즘으로 볼 수 있다.

$$Speedup = \frac{\min_{p_m \in P}\left\{\sum_{v_i \in V} w_{i,m}\right\}}{makespan} \tag{10}$$

· 알고리즘간의 우열비교

두 리스트 스케줄링 알고리즘의 makespan을 비교할 때 많은 입력 그래프에 대해서 어느정도의 그래프에 대해 더 좋은 성능을 나타내는지에 대한 비교이다. 18K개의 입력 그래프를 수행하여 어떤 알고리즘이 다른 알고리즘에 비해 makespan이 더 좋은 경우의 개수(better), 동일 한 경우(equal)의 개수, 나쁜 경우(worse)의 개수를 산출하여 비교하여 직관적인 성능 비교를 한다.

5.2 입력 그래프 생성

본 논문에서 제안한 알고리즘과 비교되는 알고리즘은 모두 이기종 프로세서 환경에서 수행되는 DAG 입력 그래프의 스케줄링을 목표로 하고 있으므로 시뮬레이션을 위해서는 적절한 이기종 환경과 많은 DAG 그래프를 생성해야 한다. 알고리즘의 공정한 성능 비교를 위해 기존의 연구에서 사용된 DAG 그래프를 인용하였으며 표준 태스크 그래프(Standard Task Graph, STDGP)[16~17] 프로젝트에서 제공하는 효율적인 표준 DAG 그래프에 아래의 매개변수를 적용하여 가중치 그래프를 생성하고 시뮬레이션을 하였다.

5.2.1 가중치 그래프 생성을 위한 매개변수

STDGP에서 제공하는 표준 DAG는 통신비용과 계산비용에 대한 정보는 없고 단지 그래프 내의 노드의 개수와 노드들의 관계만을 제공한다. 따라서 노드의 수, 통신비용, 계산비용이 적용된 DAG를 생성하기 위해 다음의 매개변수들을 입력으로 받아야만 한다.

· 통신비용 대 계산비용의 비율 (CCR). 평균 통신비용에 대한 평균 계산비용의 비율이다. 만일 CCR이 매우 낮다면 이는 연산 집약적인 응용 프로그램이라 할 수 있으며 $CCR>>1$ 이면 통신 시간이 차지하는 비율이 계산시간에 비해 상대적으로 높다는 것을 의미한다. $CCR = \{0.5, 1.0, 5.0, 10.0, 20.0\}$ 으로 설정하였다.

· 프로세서에 대한 이질성 매개변수 (β). 프로세서 속도에 대한 이질성 척도이다. 큰 값을 갖는다면 어떤 노드를 실행할 때 프로세서들 간의 계산비용에서 많은 차이를 가지게 되고 작은 값을 갖는다면 시스템상의 어떤 프로세서에 할당되어도 노드의 계산비용이 거의 같다는 것을 의미하는 매개변수이다. 그래프 상에서 각 노드 v_i의 계산비용의 평균, 즉 $\overline{w_i}$는 균일 분포를 갖는 $[0, 2\times\overline{w_{DAG}}]$범위 내에서 임의로 선택된다. 여기에서 $\overline{w_{DAG}}$는 주어진 그래프의 계산비용의 평균값으로 이는 시뮬레이션 프로그램 내에서 임의로 설정된다. 따라서 시스템 내의 각 프로세서 p_m에서 각 노드 v_i의 계산비용의 평균은 다음 범위 내에 속하게 된다.

$$\overline{w_i}\times(1-\frac{\beta}{2}) \le W_{i,j} \le \overline{w_i}\times(1+\frac{\beta}{2}) \quad (11)$$

본 논문에서는 $\beta=\{0.1,0.5,1.0,1.5,2.0\}$으로 설정하여 실험하였다.

· DAG 그래프의 노드들의 수 (v). 기존의 시뮬레이션에서 사용된 노드의 개수는 최대 140개 까지만 비교하였으나 본 논문에서는 최대 750개의 노드를 이용하여 시뮬레이션 하였다. STDGP 프로젝트에서 제공하는 표준 DAG 그래프 중에서 노드의 개수가 50, 100, 300, 500, 750인 그래프를 인용하였다.

STDGP 프로젝트에서는 노드의 수당 180개의 그래프모양을 제공한다. 따라서 매개변수 CCR과 β값을 적용하여 총 18K 개의 그래프를 생성하였고 프로세서의 수는 16개로 고정하여 시뮬레이션 하였으며 다른 매개변수를 사용하였을 경우 명시하였다.

5.3 시뮬레이션 결과

여러 가지 매개변수를 이용하여 생성된 DAG 그래프를 본 논문에서 제안한 HRPS 알고리즘과 CPOP, HCPT, FLB 알고리즘에 적용하여 시뮬레이션 하였다. 그림 4, 5는 16개의 프로세서에서 노드의 수 변화에 따른 평균 SLR과 Speedup을 보여준다. 각 노드의 개수마다 CCR과 β값을 변화시켜 3600개의 그래프를 생성하고 평균을 내었다. 평균 SLR과 Speedup 모두 HRPS, HCPT, CPOP, FLB 의 순서인 것을 확인할 수 있었으며 노드의 개수가 많을수록 다른 알고리즘 보다 더 좋은 성능이 나타나는 것을 확인할 수 있다. 전체 노드에 대해서 평균 SLR은 HRPS가 HCPT보다 13.2%, CPOP 보다 22.6%, FLB 보다 33.2%의 성능향상이 있는 것을 확인할 수 있다. 그림 6, 7은 노드의 수와 β값을 변화시켜 생성된 3600개의 그래프에 대해 CCR값의 변화에 따른 평균 SLR과 Speedup을 보여준다. 다른 알고리즘

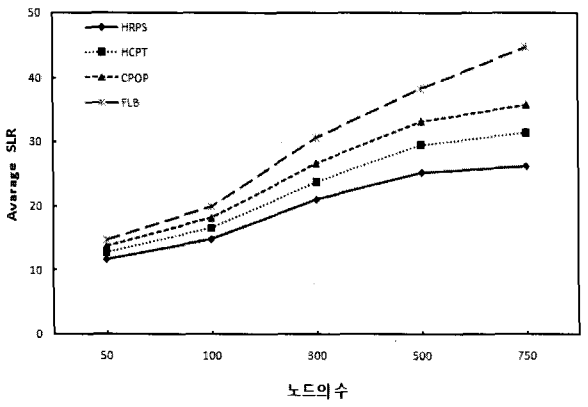


그림 4. 노드 변화에 따른 평균 SLR
Fig. 4. Average SLR for varying node.

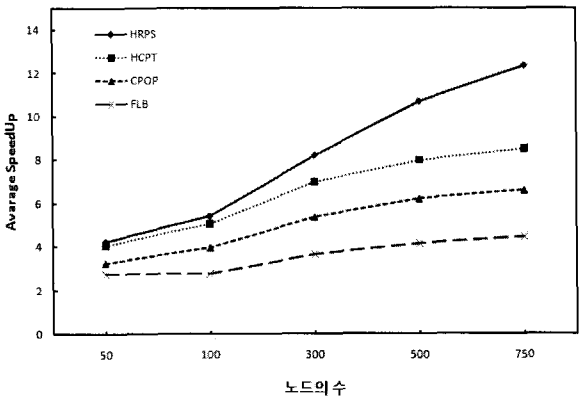


그림 5. 노드 변화에 따른 평균 SpeedUp
Fig. 5. Average SpeedUp for varying node.

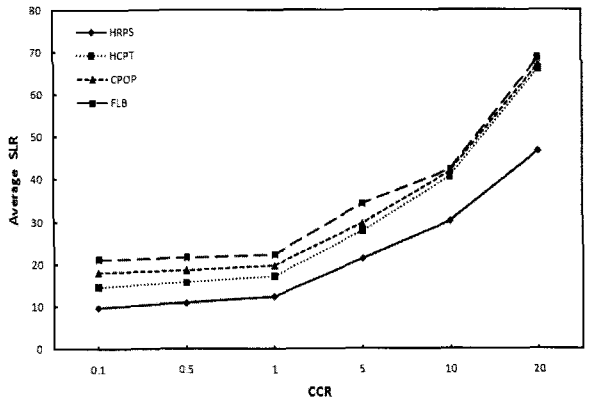


그림 6. CCR 변화에 따른 평균 SLR
Fig. 6. Average SLR for varying CCR.

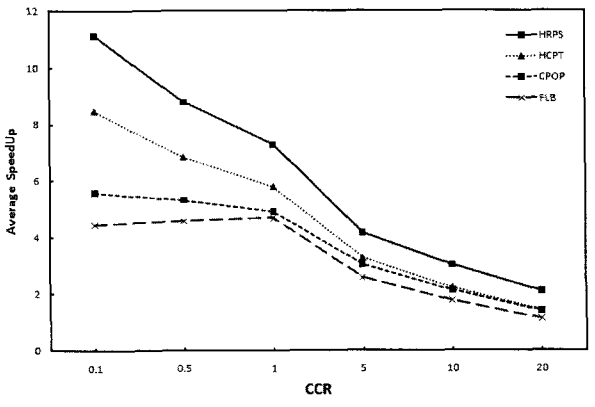


그림 7. CCR 변화에 따른 평균 SpeedUp
Fig. 7. Average SpeedUp for varying CCR.

표 3. 다른 알고리즘과 HRPS의 비교
Table 3. Comparison between HRPS and other algorithms.

		HCPT	CPOP	FLB	Combined
HRPS	Better	14418	17485	17743	91.9%
	Equal	872	70	68	1.9%
	Worse	2710	445	189	6.2%

들은 *CCR* 이 증가함에 따라 성능향상이 거의 비슷해지지만 본 논문에서 제안한 HRPS 알고리즘은 *CCR*이 증가할수록 다른 알고리즘에 비해 성능향상이 높은 것을 알 수 있다. 마지막으로 표 3은 18K개의 그래프를 시뮬레이션 했을 때 각 알고리즘 쌍의 makespan의 길이에 대한 비교를 나타낸 것이다. Combined는 better, equal, worse 에 대한 모든 값을 더해 다른 알고리즘과의 비교치를 백분율로 나타낸 것이다. 표에서와 같이 HRPS 알고리즘이 다른 알고리즘에 비해 좋은 성능을 보이고 있으며 특히 CPOP와 FLB에 대해서는 주어진 거의 모든 그래프에 대해 성능 향상이 있는 것을 확인할 수 있다.

VI. 결 론

본 논문에서는 주어진 DAG 그래프의 시작노드부터 마지막 노드까지의 경로 순위를 결정하고 순위가 높은 경로의 노드들부터 우선순위를 할당하는 새로운 리스트 스케줄링인 HRPS 알고리즘을 제안하였다. 기존의 연구에서는 임계경로만을 우선시하여 스케줄링 하였으나 본 논문에서는 시작노드부터 마지막 노드까지의 모든 경로를 고려함으로써 스케줄링의 성능을 높게 할 수 있었다. 공정한 시뮬레이션을 위해 기존의 문헌에서 연구한 DAG 그래프를 인용하여 실험하였다.

또한 기존의 실험에서 사용된 노드의 개수보다 많은 750개의 노드와 *CCR*의 크기를 20까지 하여 큰 크기의 병렬 프로그램과 확장된 이기종 환경을 고려하였다. 18K개의 DAG 그래프를 이용하여 실험한 결과 기존에 연구된 리스트 스케줄링 알고리즘보다 월등히 좋은 성능을 보였으며 특히 *CCR*이 클수록 다른 알고리즘은 모두 비슷한 성능을 보였으나 HRPS는 성능이 더 좋은 것을 확인할 수 있었다.

참 고 문 헌

[1] T.Braun, H.J. Siegel, N. Beck, L.L. Boloni, M.Maheswaran, A.I.Reuther, J.P Robertson, M.D. Theys, B.Yao, D.Hengsen, and R.F. Freund, "A Comparison Study of Static Mapping Heuristics for a Classes of Meta-Tasks on Heterogeneous Computing Systems," *Proc, Heterogeneous Computing Workshop,* pp.15-29, 1999.

[2] Oliver Sinnen, "Task Scheduling For Parallel Systems," *Wiley*, pp.7-35, 2007.

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

[4] Y.K. Kwok and I. Ahmad. "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.,* 31(4). pp. 406 - 471, 1999.

[5] Y.K Kwok and I. Ahmad. "Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors." *IEEE Trans. on Parallel and Distributed Systems,* pp. 506-521, 1996.

[6] H. Togcuglou, S, Hariri and M.Y. Wu, "Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. On Parallel and Distributed Systems*, vol 13, No.3, Feb. 2002.

[7] T. Hagras and J. Janecek, "A Simple Scheduling Heuristic for Heterogeneous Computing. Environments," *IEEE Proceedings of Second International Symposium on Parallel and Distributed Computing (ISPDC'03),* pp. 104 - 110, October 2003.

[8] A.Radulescu, and A.van Gemund, "Performance Effective and Low-complexity Task Scheduling for Heterogeneous Computing," *9th Heterogeneous Computing Workshop*, pp.229-238. 2000.

[9] T.Hagras, J.Janecek "A High Performance, Low Complexity Algorithm for Compile-Time Task Scheduling in Heterogeneous Systems," *Parallel and Computing*, 31, 653-670, 2005.

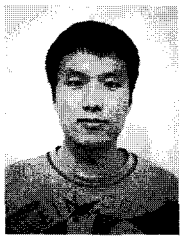
[10] S.Darbha, D.P.Agrawal. "Optimal Scheduling Algorithm for Distributed-Memory Machines," *IEEE Trans. Parallel and Distributed Systems*, 9(1), 97-95, Jan. 1998.

[11] Lan Zhou, Sun Shixin, "Scheduling algorithm based on critical tasks in heterogeneous environments," *Journal of Systems Engineering and Electronics,* Vol. 19, No. 2, pp.398-404, 2008.

[12] Kafil, M. and I. Ahmed, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, 6: 42-51, 1998.
[13] Ranaweera, A. and D.P. Agrawal, "A task duplication based algorithm for heterogeneous systems," *Proc. IPDPS*, pp: 445-450, 2000.
[14] Cristina Boeres, Jos'e Viterbo Filho and Vinod E. F. Rebello, "A cluster-based strategy for scheduling task on heterogeneous processors," *Proc. 16th Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2004.
[15] Y.K. Kwok, I. Ahmad, Benchmarking and comparison of the task graph scheduling algorithms, *J. Parallel Distrib. Comput.* 59(3). pp. 381 - 422,1999.
[16] Takao Tobita and Hironory kasahara, "A Standdart Task Graph Set for Fair Evaluation of Multiprocessor Scheduling Algorithms," *Journal of Scheduling*, 5, pp. 379-394, 2002.
[17] http://www.kasahara.elec.waseda.ac.jp.

저 자 소 개

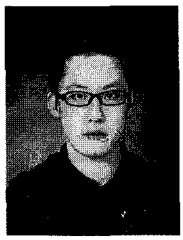
윤 완 오(학생회원)
2000년 경기대학교 전자공학과 학사 졸업.
2002년 인하대학교 전자공학과 석사 졸업.
2002년~현재 인하대학교 전자공학과 박사과정.
<주관심분야 : 분산 처리 시스템, 병렬프로그램밍, 컴퓨터 구조, 무선 통신, 컴퓨터 네트워크>



이 창 호(학생회원)
2008년 청주대학교 전자공학과 학사 졸업.
2008년~현재 인하대학교 석사과정.
<주관심분야 : 병렬 및 분산 처리 시스템, 컴퓨터구조, Fault-tolerant computing>

최 상 방(평생회원)
1981년 한양대학교 전자공학과 학사 졸업.
1981년~1986년 LG 정보통신(주)
1988년 University of washinton 석사 졸업.
1990년 University of washinton 박사 졸업.
1991년~현재 인하대학교 전자공학과 교수
<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, Fault-tolerant computing>

윤 정 희(학생회원)
1998년 인하대학교 전자계산공학과 학사 졸업.
2006년 인하대학교 정보컴퓨터교육학과 석사 졸업.
2008년~현재 인하대학교 전자공학과 박사과정.
<주관심분야 : 컴퓨터 아키텍처, 병렬프로그래밍>



김 효 기(학생회원)
2008년 인하대학교 전자공학과 학사 졸업.
2008년~현재 인하대학교 석사과정.
<주관심분야 : 분산 처리 시스템, 병렬프로그램밍>