분산 이기종 시스템에서 리스트 스케줄링 알고리즘을 위한 새로운 프로세서 할당 정책

(A Novel Processor Allocation Policy for List Scheduling in Distributed Heterogeneous Computing System)

윤 완 오 † 송 인 성 † 윤 준 철 † 최 상 방 ††
(Wan-Oh Yoon) (In-Seong Song) (Jun-Chol Yoon) (Sang-Bang Choi)

요 약 분산 이기종 시스템의 성능은 DAG로 주어지는 입력 그래프를 스케줄링 하는 알고리즘의 성능에 좌우된다. 많은 스케줄링 알고리즘 중에 리스트 스케줄링 알고리즘은 낮은 복잡도를 가지면서 우수한 성능을 보이고 있다. 리스트 스케줄링은 태스크 우선순위 결정 단계와 프로세서 할당 단계로 이루어져 있으나 대부분의 연구들은 태스크 우선순위 결정 단계만을 연구하고 있다. 본 논문에서는 기존의 할당 정책과 동일한 복잡도를 가지면서 성능이 향상된 새로운 프로세서 할당 정책인 LIP 정책을 제안한다. 기존의 리스트 스케줄링 알고리즘인 HEFT, HCPT, GCA, PETS의 태스크 우선순위 결정 정책에 본 논문에서 제안한 LIP 정책을 적용하여 실험한 결과 기존의 프로세서 할당 정책인 삽입 정책과 비 삽입 정책보다 성능 향상이 있는 것을 확인할 수 있었다.

키워드 : 분산시스템, DAG, 리스트 스케줄링, 프로세서 할당 정책, 이기종 시스템

Abstract The performance of Distributed Heterogeneous Computing System depends on the algorithm which schedules input DAG graph. Among various scheduling algorithms, list scheduling algorithm provides superior performance with low complexity. List scheduling consists of task prioritizing phase and processor allocation phase, but most studies only focus on task prioritizing phase. In this paper, we propose LIP policy which has the same complexity with traditional allocation policies but has superior performance. The performance of LIP has been observed by applying them to task prioritizing phase of traditional list scheduling algorithms, HCPT, HEFT, GCA, and PETS. The results show that LIP has better performance than insertion-based policy and non-insertion-based policy, which are traditional processor allocation policies.

Key words : Distributed System, DAG, list scheduling, processor allocation policy, heterogeneous system

1. 서 론

최근 네트워크 기술의 발달로 서로 다른 성능을 갖는 프로세서들 간에 효율적인 통신이 가능하게 되면서 고속연산이 절실히 요구되는 대용량의 병렬 프로그램을 처리할 수 있는 분산 이기종 시스템(Distributed Heterogeneous Computing System, DHCS)의 연구가 활발히 이루어지고 있다[1]. DHCS의 성능 향상은 병렬 프로그램을 프로세서에 할당하여 프로그램의 전체 실행 시간을 최소화하는 스케줄링 알고리즘과 연관되어 있다. 최적의 스케줄링을 통해 DHCS의 성능 향상을 꾀할 수 있지만 이는 다항 시간(polynomial time)내에 결과를 얻을 수 없는 NP-complete 문제이다[2-5]. 따라서 많

† 학생회원 : 인하대학교 전자공학과
wanoh38@gmail.com
nicvirus@inha.edu
wnscjf78@paran.com
†† 종신회원 : 인하대학교 전자공학과 교수
sangbang@inha.ac.kr
논문접수 : 2009년 12월 7일
심사완료 : 2009년 12월 30일

정보과학회논문지: 시스템 및 이론 제37권 제2호(2010.4)

은 연구들이 만족할만한 시간 복잡도(time complexity)와 최적에 가까운 해결책을 얻기 위해 NP-complete 문제를 해결하는데 중점을 두고 있다. 이러한 해결책은 스케줄링을 실행하는 시점에 따라 정적, 동적으로 나눌 수 있다. 정적 스케줄링은 각 태스크의 계산비용(computation cost)이나 태스크들 사이의 통신비용(communication cost)과 같이 스케줄링에 필요한 시스템 정보를 미리 알고 병렬 프로그램이 실행되기 전 컴파일(compile) 시간에 스케줄링을 실행하는 것으로 실행 시간(run time)중에는 오버헤드(overhead)가 발생하지 않는다. 이와는 반대로 동적 스케줄링은 병렬 프로그램이 실행되는 동안에 스케줄링에 필요한 정보를 획득하여 스케줄링 하는 방법이다. 정적 스케줄링은 복제(duplication) 스케줄링, 클러스터(cluster) 스케줄링, 리스트(list) 스케줄링과 임의 탐색 기반(guided random search) 스케줄링 알고리즘으로 분류할 수 있다.

리스트 스케줄링은 크게 두 단계로 이루어져 있다. 첫 번째 단계는 태스크 간의 상호 제약을 충족시키면서 특정 우선순위 함수에 기초하여 태스크의 우선순위를 결정하는 우선순위 결정 단계이며, 두 번째 단계는 최우선 태스크부터 적절한 프로세서를 선택하여 할당하는 프로세서 할당단계이다. 리스트 스케줄링은 다른 분류의 알고리즘에 비해 비교적 낮은 복잡도를 가지면서 효율적인 성능을 보여준다. 대표적인 알고리즘으로는 HCPT (Heterogeneous Critical Parent Trees)[6], HRPS(Heterogeneous Rank-Path Scheduling)[7], HEFT(Heterogeneous Earliest Finish Time)[8], CPOP(Critical Path On a Processor)[8], GCA(Generalized Critical-task Anticipation)[9], LMT(Levelized Min Time)[10], PETS(Performance Effective Task Scheduling algorithm)[11] 그리고 HPS(High Performance-task Scheduling)[12] 알고리즘이 있다. 이와 같은 알고리즘들은 태스크의 우선순위를 결정하는 우선순위 함수는 모두 다르지만 프로세서 할당 단계에서는 비 삽입 기반(non-insertion-based) 정책을 사용하는 HCPT 알고리즘을 제외하고 모두 삽입 기반(insertion-based) 정책을 사용하고 있다. 두 정책 모두 현재 할당할 태스크의 실행 완료 시간이 최소가 되도록 프로세서를 할당하는 정책이다.

본 논문에서는 기존의 삽입 기반 정책과 동일한 복잡도를 가지면서 효율적으로 프로세서를 할당하는 LIP (Look-ahead Insertion Policy)를 제안한다. LIP는 우선순위가 결정된 태스크의 순서에 따라 현재 할당될 태스크와 다음에 할당될 태스크의 관계를 고려한 후 두 태스크 사이의 통신비용과 계산비용을 고려하여 다음 태스크의 실행 완료 시간이 최소가 되도록 현재의 태스크를 할당하는 정책이다. 다음에 할당될 태스크의 실행 완료 시간이 최소가 되도록 현재의 태스크를 할당함으로써 전체 실행 시간을 최소화시킬 수 있다. 제안하는 LIP 정책의 성능을 확인하기 위해 여러 가지 매개변수를 적용하여 만들어진 입력 그래프를 기존의 리스트 스케줄링 알고리즘인 HEFT, GCA, PETS 그리고 HCPT에서 사용된 우선순위 함수를 이용하여 태스크의 우선순위를 결정하고 LIP 정책을 적용한 알고리즘과 원래의 알고리즘과의 성능 비교를 하였다. 비교한 결과 본 논문에서 제안한 LIP 정책을 적용한 알고리즘이 기존의 삽입 정책과 비 삽입 정책을 적용한 알고리즘보다 우수한 것을 확인할 수 있었다.

본 논문은 다음과 같이 이루어져 있다. 2장에서는 이기종 분산 시스템 환경과 병렬 프로그램 및 매개변수에 대해 설명하고 3장에서는 본 논문의 시뮬레이션에서 비교된 기존의 대표적인 리스트 스케줄링 알고리즘인 HCPT, HEFT, GCA, PETS에 대해 설명한다. 4장에서는 본 논문에서 제안하는 프로세서 할당 정책인 LIP를 소개하고 5장에서는 다양한 입력 그래프와 DHCS 환경을 이용한 실험을 통해 제안하는 프로세서 할당 정책의 성능을 분석한다. 마지막으로 6장에서 결론을 맺는다.

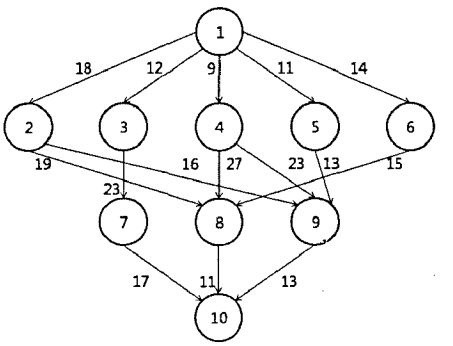

그림 1 10개의 노드를 갖는 DAG

표 1 계산비용 행렬(W)와 평균 계산비용($\overline{w_i}$)

$node$	p_1	p_2	p_3	$\overline{w_i}$
v_s	14	16	9	13
v_2	13	19	18	16.67
v_3	11	13	19	14.33
v_4	13	8	17	12.67
v_5	12	13	10	11.67
v_6	13	16	9	12.67
v_7	7	15	11	11
v_8	5	11	14	10
v_9	18	12	20	16.67
v_e	21	7	16	14.67

2. 스케줄링을 위한 문제 정의

이 장에서는 리스트 스케줄링 알고리즘을 적용하기 위해 병렬 프로그램을 모델링한 그래프와 DHCS 환경 그리고 본 논문에서 제안하는 알고리즘을 위한 매개변수에 대해 설명한다. DHCS 환경에서 리스트 스케줄링 알고리즘을 위해 입력으로 주어지는 병렬 프로그램은 그림 1과 같이 노드(node) 사이에 상호 제약이 존재하며 방향성 비순환 그래프(DAG)로 표현된다. 방향성 비순환 그래프 G는 (V,E)로 모델링이 가능하며 여기에서 $V=\{v_1,v_2,...,v_n\}$는 병렬 프로그램 내에 n개의 태스크를 노드로 표현한 v의 집합이고 노드들 중에서 i번째 노드는 v_i로 표현한다. 본 논문에서 노드와 태스크는 동일한 의미로 사용된다. E는 두 노드 사이의 관계를 표현하는 에지(edge) e의 집합으로 에지 $e_{i,j}$는 노드 v_i에서 노드 v_j로의 연결이며 이는 노드 v_j가 실행되기 전 노드 v_i의 실행이 완료되어야 한다는 것을 의미한다. 이 경우 v_i을 v_j의 부모노드(parent node)라고 부르고, v_j는 v_i의 자식노드(child node)라고 부른다. 또한 노드 v_i의 모든 부모노드의 집합을 $pred(v_i)$, 모든 자식노드의 집합을 $succ(v_i)$로 정의한다. 그림 1의 노드 1처럼 부모노드가 존재하지 않는 노드를 시작노드(start node, v_s)라 부르고 노드 10처럼 자식노드가 없는 노드를 출력노드(exit node, v_e)라 부른다. 리스트 스케줄링을 위한 DAG 그래프는 오직 한 개의 시작노드와 출력노드를 가져야 한다. 만약 2개 이상의 입력노드와 출력노드가 있다면 이 노드들을 묶을 수 있는 의사노드(pseudo node)를 추가할 수 있으며 의사노드가 프로세서에서 수행될 때의 계산비용과 통신비용은 모두 0값을 가지고 이 노드는 스케줄링 상에 아무런 영향을 주지 않는다고 가정한다.

DHCS는 m개의 이기종 프로세서 p들이 완전연결(fully connected)로 구성된 집합 P로 나타내고 각 프로세서 사이의 통신채널 및 대역은 충분하다고 가정한다. 또한 각 프로세서는 노드의 수행과 통신이 동시에 이루어지며 노드가 수행하는 동안 어떠한 방해없이 수행이 가능하고 수행결과는 즉시 다른 프로세서에 보낼 수 있다고 가정한다.

W는 표 1과 같이 노드 $v_i \in V$가 임의의 프로세서 $p_k \in P$에서 실행될 때의 계산비용 $w_{i,k}$로 이루어진 $|V|\times|P|$ 크기의 계산비용 행렬이다. 스케줄링 알고리즘을 수행하기 전에 각 노드의 평균 계산비용을 계산해야 한다. 예를 들어 m개의 프로세서에서 노드 v_i 의 평균 계산비용 $\overline{w_i}$는 식 (1)을 이용하여 계산한다.

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

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

$$cmc_{(i,k)(j,m)} = \begin{cases} 0 & \text{if } p_k = p_m \\ S_k + R_{k,m} \cdot \mu_{i,j} & otherwise \end{cases} \tag{2}$$

여기서 S_k는 프로세서 p_k의 통신 시작 시간(sec), $\mu_{i,j}$는 노드 v_i부터 노드 v_j로 전달되는 데이터 전송 양(bytes) 그리고 $R_{k,m}$은 프로세서 p_k부터 p_m까지 바이트 당 통신비용(sec/byte)이다. 프로세서 p_k와 p_m이 같은 프로세서일 경우 통신비용은 0값을 가지며 서로 다른 프로세서라면 통신비용은 통신 시작 시간과 μ 바이트 크기의 데이터가 전달되는데 필요한 시간의 합으로 나타낼 수 있다. 그림 1의 DAG에서 에지 $e_{i,j}\in E$에 표현된 수치는 평균 통신비용 $\overline{cmc_{i,j}}$을 식 (3)을 이용하여 구해진 값을 표현한 것이다. 여기서 $\overline{S}$는 모든 프로세서의 평균 통신 시작 시간이며 $\overline{R}$은 모든 프로세서에 대한 바이트 당 평균 통신비용이다.

$$\overline{cmc_{i,j}} = \overline{S} + \overline{R} \cdot \mu_{i,j} \tag{3}$$

$$where, \begin{cases} \overline{S} = \sum_{k=1}^{m} \frac{S_k}{m} \\ \overline{R} = \sum_{k=1}^{m}\sum_{l=1}^{m} \frac{R_{k,l}}{(m(m-1))} \end{cases}$$

그림 2에서와 같이 어떤 노드 v_i부터 출력노드 v_e까지 가장 긴 경로를 $b_level(v_i)$로 정의 하고 식 (4)를 이용하여 출력노드부터 재귀적으로 계산한다. 출력노드의 $b\_level(v_e) = \overline{w_e}$ 로 정의된다.

$$b\_level(v_i) = \overline{w_i} + \max_{v_j \in succ(v_i)} \{ cmc_{i,j} + b\_level(v_j) \} \tag{4}$$

마찬가지로 노드 v_i부터 시작노드 v_s까지 가장 긴 경로를 $t_level(v_i)$로 정의하고 다음 식을 이용하여 시작노드부터 재귀적으로 계산한다.

$$t\_level(v_i) = \max_{v_j \in pred(v_i)} \{ t\_level(v_j) + \overline{w_j} + cmc_{j,i} \} \tag{5}$$

임계경로(critical path)는 시작노드부터 출력노드까지의 가장 긴 경로를 의미한다. 노드 v_i가 프로세서 p_k에서 실행될 수 있는 초기 시작 시간(earliest start time),

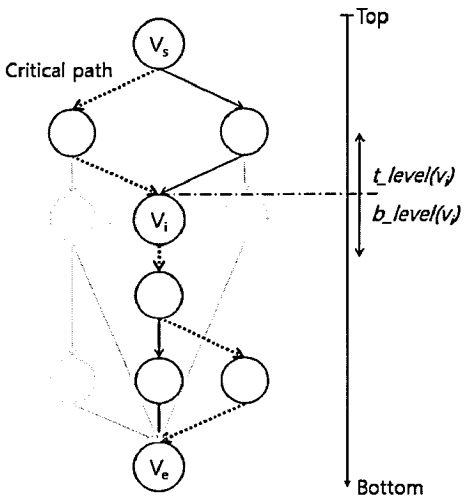

그림 2 노드 v_i의 $t_level(v_i)$과 $b\_level(v_i)$

$est(v_i, p_k)$을 식 (6)과 같이 정의하고 노드가 프로세서에서 실행이 완료되는 실행 완료 시간(earliest completion time), $ect(v_i, p_k)$을 식 (7)과 같이 정의한다.

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

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

여기에서 $pred(v_i)$는 노드 v_i의 부모노드의 집합을 의미하며 $PA[p_k]$는 프로세서 p_k에서 노드를 실행할 수 있는 실행 가능 시간을 의미한다. 처음 시작노드가 프로세서 p_k에서 실행될 경우 $est(v_s, p_k) = 0$의 값을 가지게 된다. $\max(ect(v_j) + k \cdot cmc_{j,i})$는 노드 v_i가 프로세서 p_k에서 부모노드들 중에서 가장 늦게 데이터를 전달받는 시간을 의미하며 $LDRT(v_i, p_k)$(Latest Data Receive Time)으로 정의된다. 또한 가장 늦게 데이터를 전달하는 부모노드를 임계부모노드(Critical Immediate Parent-Node, CIP)라 한다. 여기서 상수 k는 부모노드가 같은 프로세서에 할당되었을 경우 0의 값을 가지고 그 외에는 1의 값을 가지게 된다. 노드의 실행 완료 시간 ect는 노드의 시작 시간에 실행 시간을 더한 값으로 표현되며 ect 중에서 최소의 값을 갖는 ect을 min_ect로 정의한다. 입력으로 주어지는 DAG의 전체 실행 시간은 $makespan$으로 정의하며 식 (8)과 같이 출력노드 v_e의 min_ect으로 나타낼 수 있다.

$$makespan = min\_ect(v_e, p_m) \quad (8)$$

본 논문에서 제안한 프로세서 할당 정책은 복잡도를 줄이면서 $makespan$을 최소화하는데 그 목적이 있다.

3. 관련 연구

지금까지 연구된 리스트 스케줄링 알고리즘은 여러 가지가 있으나 이 장에서는 본 논문에서 제안하는 프로세서 할당 정책의 성능 검증을 위해 사용된 리스트 스케줄링 알고리즘인 HCPT, HEFT, GCA, PETS에 대해 설명한다.

3.1 HCPT(Heterogeneous Critical Parent Trees)

HCPT 알고리즘은 DAG의 임계경로를 이루는 노드인 임계노드(critical node)라는 개념을 사용하는 알고리즘으로 다른 리스트 스케줄링 알고리즘과 같이 두 단계로 이루어진다. 우선 첫 번째 단계인 우선순위 결정 단계에서는 DAG의 임계경로를 구하고 비어있는 큐(queue) L과 임계경로를 이루는 임계노드들이 저장된 스택 S를 갖고 시작한다. 다음으로 스택 S의 최상위에 있는 노드의 부모노드가 큐에 있는지 확인한 후 큐에 부모노드가 없다면 부모노드를 스택에 *push*하고 큐에 부모노드가 있다면 스택의 최상위 노드를 *pop*하여 큐에 넣는다. 스택에 있는 모든 임계노드가 없어질 때까지 이와 같은 동작을 반복하면 최종적으로 큐에 저장된 순서대로 노드의 우선순위가 결정된다. 두 번째 단계에서는 큐에 있는 노드를 하나씩 선택하여 가장 빠른 시간에 노드의 계산을 끝낼 수 있는 프로세서에 노드를 할당한다. 이 알고리즘의 시간 복잡도는 $O(pv^2)$이다. 여기서 p는 프로세서, v는 노드를 의미한다.

3.2 HEFT(Heterogeneous Earliest Finish Time)

HEFT 알고리즘은 노드 우선순위 결정 단계, 프로세서 할당 단계의 두 단계로 이루어진 알고리즘이다. 첫 번째 단계인 우선순위 결정 단계에서는 모든 노드의 b_level 값을 계산하고 내림차순으로 정렬하여 노드의 우선순위를 결정한다. 두 번째 단계인 프로세서 할당 단계에서는 삽입 기반 정책을 이용하여 노드를 할당한다. 삽입 기반 정책은 이미 할당된 두 노드 사이에 다른 노드를 할당할 공간이 있고, 선행 제약 조건을 위반하지 않는다면 빈 공간에 노드를 추가로 할당하는 방법이다. 이 알고리즘의 시간 복잡도는 $O(pv^2)$이다.

3.3 GCA(Generalized Critical-task Anticipation)

GCA 알고리즘은 두 단계로 이루어져 있으며 두 번째 단계인 프로세서 할당 단계는 HEFT 알고리즘과 마찬가지로 삽입 정책을 이용하여 최소의 실행 시간을 갖는 프로세서에 노드를 할당한다. 첫 번째 단계인 노드 우선순위 결정 단계에서 우선 각 노드의 우선순위 값을 b_level을 이용하여 계산한다. 이 알고리즘이 기존의 알고리즘인 HEFT 알고리즘과 다른 점은 우선순위 값을 내림차순으로 정렬하여 노드의 우선순위를 결정하지 않고 자식노드에 가장 늦게 데이터를 전달하는 노드의 우

선순위를 높게 하는 방법을 사용한다는 점이다. 이 알고리즘의 시간 복잡도는 $O(pv^2)$이다.

3.4 PETS(Performance Effective Task Scheduling)

PETS 알고리즘은 두 단계로 이루어진 기존의 리스트 스케줄링 알고리즘에 레벨 정렬 단계를 추가한 리스트 스케줄링 알고리즘으로 세 단계로 이루어져 있다. 첫 번째 단계인 레벨 정렬 단계에서는 입력 그래프 DAG의 각 레벨에 속한 노드들을 서로 독립인 노드들끼리 그룹화 한다. 두 번째 단계인 노드 우선순위 결정 단계에서는 각 레벨의 노드에 우선순위 함수를 적용하여 우선순위 값에 따라 노드의 스케줄링 순서를 결정한다. 우선순위 함수는 노드의 평균 계산비용, 자식노드의 통신비용의 합과 부모노드 중에서 가장 큰 우선순위 값의 합으로 구성된다. 세 번째 단계인 프로세서 할당 단계에서는 두 번째 단계에서 결정된 노드의 순서에 따라 삽입 기반 정책을 이용하여 최소 실행 시간을 제공하는 프로세서에 노드를 할당한다. 이 알고리즘의 시간 복잡도는 $O(v+e)(p+\log v)$이다.

4. 제안하는 스케줄링 알고리즘

기존의 연구에서 프로세서를 선택하여 노드를 할당하기 위한 방법은 크게 삽입 기반 정책과 비 삽입 기반 정책으로 나눌 수 있다. 비 삽입 기반 정책은 선행 제약을 만족하면서 최소의 실행 완료 시간을 제공하는 프로세서에 노드를 할당하는 정책이고 삽입 기반 정책은 이미 할당되어진 두 노드 사이에 새로운 노드를 할당할 수 있는 여유 공간이 있고 선행 제약을 만족한다면 노드의 삽입까지 고려해서 *ect*가 최소인 프로세서에 할당하는 정책이다. 삽입 기반 정책의 성능을 증가시키기 위해서는 기반이 되는 비 삽입 기반 정책의 효율성이 증가되어야 한다. 그림 3은 비 삽입 기반 정책의 문제점을 보여주고 있다.

그림 3의 (a), (b)와 같은 입력 그래프와 3개의 프로세서에 대한 각 노드의 실행 시간이 주어졌을 경우에 지금까지 제안된 리스트 스케줄링 알고리즘 중에서 어떤 노드 우선순위 함수를 이용하더라도 노드의 순위는 A-B-C로 결정된다. 정해진 우선순위에 따라 노드를 프로세서에 할당하기 위해 삽입 정책을 사용한다면 A노드는 최소의 실행 완료 시간 2를 제공하는 프로세서 p_1에 B와 C노드는 프로세서 p_3에 할당되어 그림 3의 (c)와 같이 전체실행 시간이 15인 결과를 얻을 수 있으나 이것은 최선의 결과 값이 아니다. 만일 그림 3의 (d)와 같이 A노드의 최소 완료 시간이 아닌 다음에 할당될 노드인 B의 실행 완료 시간이 최소가 되도록 B의 부모노드인 A노드를 p_3 프로세서에 할당한다면 11의 결과를 얻을 수 있기 때문이다. 즉 현재노드가 자식노드에게 데이터를 가장 늦게 전달하는 임계부모노드이고 자식노드와의 통신비용이 크다면 부모노드와 자식노드를 같은 프로세서에 할당하는 것이 자식노드의 실행 완료 시간을 줄일 수 있다. 하지만 부모-자식 두 노드를 최적의 프로세서에 할당하기 위해 모든 프로세서에 대해서 최적의 프로세서를 찾게 되면 복잡도가 증가되는 문제점이 있다. 따라서 본 논문에서는 기존의 삽입 기반 정책과 동일한 복잡도를 가지면서 기존의 프로세서 할당 정책보다 향상된 성능을 가지는 LIP(Look-ahead Insertion Policy)를 제안한다. 우선순위가 결정된 노드의 순서에 따라 현재 할당될 노드와 다음에 할당될 노드의 관계를 고려한 후 두 노드 사이의 통신비용과 계산비용을 고려하여 다음 노드의 실행 완료 시간이 최소가 되도록 현재의 노드를 할당하는 LIP 정책은 두 단계로 이루어져 있다. 첫 번째 단계는 우선순위가 결정되어 있는 노드들을 부모노드와 자식노드로 이루어진 그룹으로 나누는 그룹화 단계이고 두 번째 단계는 그룹화된 노드들을 최적의 프로세서에 할당하는 단계이다.

그룹화 단계에서는 리스트 순서에 따라 그림 4와 같이 최대 2개의 노드를 가지면서 부모-자식노드로 구성되는 세부 그룹으로 그룹화를 한다. 예를 들어 그림 1의 DAG와 표 1처럼 3개 프로세서에 대한 각 노드의 실행

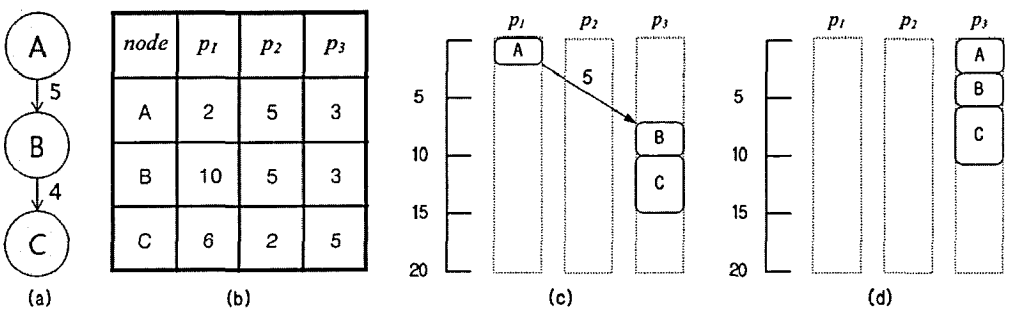

그림 3 (a) (b) 비 삽입 정책의 문제점에 대한 DAG와 실행시간 예제, (c) (d) 프로세서 할당 예

```
G0 = {vs}
i=0
for each node except vs in the list  // O(v)
        if current node has a dependency with a node in Gi and ||Gi||<2
                add node to Gi
        else
                i++
                create an empty group Gi
                add node to Gi
        end if
end for
```

그림 4 그룹화 단계의 의사 코드

시간이 주어지고 기존의 우선순위 알고리즘을 적용하여 리스트 L에 $\{v_s, v_2, v_3, v_7, v_4, v_5, v_9, v_6, v_8, v_e\}$의 순서대로 노드의 우선순위가 결정되어 있다고 가정한다. 리스트 L의 노드들을 세부 그룹으로 그룹화하기 위해서 우선 새로운 그룹 G_0을 만들고 노드 v_s을 G_0에 추가한다. 다음 노드 v_2는 그룹 G_0에 속한 노드 v_s의 자식노드이므로 그룹 G_0에 추가한다. G_0의 노드 개수가 2개이므로 새로운 그룹 G_1을 만들고 노드 v_3을 추가한다. v_7은 v_3의 자식노드이므로 G_1에 추가되고 G_1의 노드 개수가 2개이므로 새로운 그룹 G_2을 만들고 v_4을 추가한다. 노드 v_5는 v_4와 부모-자식 관계가 없으므로 새로운 그룹 G_3을 만들고 G_3에 추가한다. 이와 같은 방법으로 리스트 L의 노드들을 그룹화 하면 $G_0 = \{v_s, v_2\}$, $G_1 = \{v_3, v_7\}$, $G_2 = \{v_4\}$, $G_3 = \{v_5, v_9\}$, $G_4 = \{v_6, v_8\}$, $G_5 = \{v_e\}$의 결과를 얻게 된다.

다음은 그림 5와 같이 그룹화된 노드들을 자식노드의 실행 시간이 최소가 되도록 부모노드를 최적의 프로세서에 할당하는 단계이다. i번째 그룹 내에 있는 두 개의 노드 중 첫 번째 노드를 G_i_FN(First Node), 두 번째 노드를 G_i_SN(Second Node)로 정의한다. 노드의 개수가 두 개인 그룹에 대해서 삽입 기반 정책을 이용하여 G_i_FN의 최소 완료 시간(min_ect)을 제공하는 프로세서 p_k을 찾고 G_i_FN이 p_k에 할당되었다고 가정한 후, 두 번째 노드 G_i_SN의 min_ect을 제공하는 프로세서 p_l을 찾는다. 이때의 min_ect을 ect_1이라 정의한다. $p_k \neq p_l$임과 동시에 G_i_FN이 G_i_SN에게 데이터를 가장 늦게 주는 임계부모노드라면 G_i_FN이 프로세서 p_l에 할당되었다 가정하고 p_l에서 G_i_SN의 ect을 다시 계산

```
i=0
while there are unscheduled nodes in the Gi  // O(v)
        if(||Gi|| == 2)
                for each processor pm in the processor set pm∈P  // O(p)
                        compute ect(Gi_FN, pm) value using insertion-based policy  // O(v)
                end for
                assume assigning node Gi_FN to the processor pk that gives min_ect of node Gi_FN
                for each processor pm in the processor set pm∈P  // O(p)
                        compute ect(Gi_SN,pl) value using insertion-based policy  // O(v)
                end for
                assume assigning node Gi_SN to the processor pl that gives min_ect of node Gi_SN
                ect1 = ect(Gi_SN, pl)
                if(pk ≠ pl and Gi_FN∈ CIP(Gi_SN))
                        assume assigning node Gi_FN to the processor pl
                        compute ect(Gi_SN, pl)
                        ect2 = ect(Gi_SN, pl)
                        if(ect1 < ect2)
                                assign node Gi_FN to the processor pk
                                assign node Gi_SN to the processor pl
                        else
                                assign node Gi_FN to the processor pl
                                assign node Gi_SN to the processor pl
                        endif
                endif
        else
                for each processor pm in the processor set pm∈P  // O(p)
                        compute ect(Gi_FN, pm) value using insertion-based policy  // O(v)
                end for
                assign node Gi_FN to the processor pk that gives min_ect of node Gi_FN
        endif
        i++
endwhile
```

그림 5 제안하는 프로세서 할당 정책의 의사 코드

한다. 이때의 ect을 ect_2라고 정의한다. 마지막으로 ect_1과 ect_2을 비교하여 작은 ect을 만족하는 프로세서에 첫 번째 노드인 G_i_FN을 할당하게 된다. 앞에서 가정한 리스트 L의 $\{v_s, v_2, v_3, v_7, v_4, v_5, v_9, v_6, v_8, v_e\}$로 우선순위가 결정된 노드들을 프로세서에 할당하는 과정을 아래의 표 2에 나타내었다. 그룹 G_0의 두 노드 v_s, v_2에서 v_s의 min_ect을 제공하는 프로세서는 p_3이다. p_3에 v_s가 할당되었다고 가정하고 v_2의 min_ect을 계산하면 v_2의 min_ect을 제공하는 프로세서 역시 v_s와 같은 p_3이기 때문에 두 노드는 모두 p_3에 할당된다.

G_1의 v_3와 v_7 역시 동일한 프로세서 p_1에서 min_ect 값을 갖기 때문에 모두 p_1에 할당되고 그룹 G_2에는 v_4 노드 하나밖에 없기 때문에 min_ect을 만족하는 프로세서 p_2에 할당되게 된다. G_3의 v_5는 p_3에서 37의 값을 갖게 되고 이때의 v_9는 p_2에서 62의 값을 갖는다. v_5와 v_9는 서로 다른 프로세서에서 min_ect 값을 가지고 v_9의 임계부모노드가 v_5이기 때문에 v_5을 프로세서 p_3에서 p_2로 이동하여 할당하고 프로세서 p_2에서 v_9의 ect 값을 다시 계산하면 전체 실행 시간은 55가 된다. 따라서 v_5와 v_9는 모두 프로세서 p_2에 할당된다. 마찬가지로 G_4의 노드 v_6와 v_8도 프로세서 p_1에 함께 할당되고 전체 실행 시간은 76이 된다. 그림 6은 $\{v_s, v_2, v_3, v_7, v_4, v_5, v_9, v_6, v_8, v_e\}$의 순서대로 결정된 노드들을 비 삽입 정책, 삽입 정책 그리고 본 논문에서 제안하는 LIP를

표 2 프로세서 할당 과정

group	node	processor						ect_1	ect_2	processor selected
		p_1		p_2		p_3				
		est	ect	est	ect	est	ect			
G_0	v_s	0	14	0	16	0	9	-	-	p_3
	v_2	27	40	27	46	9	27	27	27	p_3
G_1	v_3	21	32	21	34	27	46	-	-	p_1
	v_7	32	39	55	70	55	66	39	39	p_1
G_2	v_4	39	52	18	26	27	44	-	-	p_2
G_3	v_5	39	51	26	39	27	37	-	-	$p_3 \rightarrow p_2$
	v_9	50	68	50	62	49	69	62	-	p_2
	v_9	-	-	43	55	-	-	-	55	p_2
G_4	v_6	39	52	55	71	27	36	-	-	p_3
	v_8	53	58	55	66	53	67	-	-	p_1
G_5	v_e	68	89	69	76	69	85	-	-	p_2

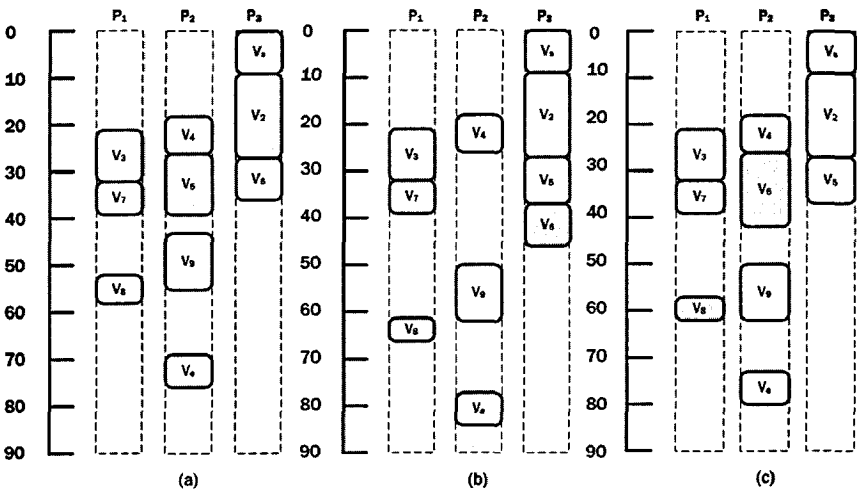

그림 6 (a) LIP 기반 할당 정책, (b) 비 삽입 기반 할당 정책, (c) 삽입 기반 할당 정책

사용하여 프로세서에 할당한 결과를 보여 주고 있다. 비 삽입 정책은 84, 삽입 정책은 80으로 LIP의 76이 제일 우수한 성능을 보임을 알 수 있다.

시간 복잡도를 살펴보면 세부 그룹화 단계에서 $O(v)$의 시간이 소요되며, 프로세서 할당 과정에서 각 노드에 대한 검사 및 ect 비교에 $O(2pv^2)$의 시간이 소요되어 총 $O(pv^2)$의 시간이 소요된다.

5. 성능 분석 및 평가

이 장에서는 HCPT, HEFT, GCA, PETS 알고리즘을 이용하여 본 논문에서 제안하는 새로운 프로세서 할당 정책인 LIP의 성능을 실험을 통해 분석한다. HCPT, HEFT, GCA, PETS 알고리즘에서 제안한 노드 우선순위 결정 방법은 그대로 사용하고 프로세서 할당 정책에서 사용한 삽입 정책과 비 삽입 정책대신 본 논문에서 제안한 LIP 정책을 적용한 알고리즘을 HCPT_LIP, HEFT_LIP, GCA_LIP, PETS_LIP로 정의하여 기존의 알고리즘과 성능 분석을 하였다.

5.1 성능 비교 기준

알고리즘의 성능 비교를 위해서 다음과 같은 기준을 사용하였다.

• Makespan

makespan은 주어진 입력 그래프 DAG의 실행이 완료되는데 소요된 총 시간을 의미하며 출력노드 v_e의 실행이 완료된 시간으로 표현할 수 있다.

$$makespan = ect(v_e, p_m) \tag{9}$$

• Schedule Length Ratio(SLR)

리스트 스케줄링 알고리즘 성능 측정의 주 기준으로는 makespan이 사용된다. 하지만 각각 다른 속성을 가진 많은 양의 DAG들이 이용되기 때문에, Schedule Length Ratio(SLR)라 불리는 하한(lower bound) 값으로 makespan을 표준화 시켜야한다. SLR은 다음 식으로 정의할 수 있다.

$$SLR = \frac{makespan}{\sum_{v_i \in CP} \min_{p_m \in P}\{w_{i,m}\}} \tag{10}$$

분모는 주어진 DAG 그래프에서 임계경로를 이루는 노드들의 최소 계산비용의 합이다. 하한 값을 분모로 이용하기 때문에 어떤 알고리즘을 이용하더라도 알고리즘의 SLR은 1보다 작아질 수 없다. 따라서 SLR 값이 작을수록 알고리즘의 성능이 좋다고 할 수 있다.

• Speedup

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

$$Speedup = \frac{\min_{p_m \in P}\left\{\sum_{v_i \in V} w_{i,m}\right\}}{makespan} \tag{11}$$

• 프로세서 할당 정책 간의 우열 비교

두 리스트 스케줄링 알고리즘의 makespan을 비교할 때 많은 양의 입력 그래프를 사용하여 주어진 특성을 갖는 그래프에 대해 어느 프로세서 할당 정책이 더 좋은 성능을 나타내는지를 비교한다. 본 논문에서는 22.5K개의 입력 그래프를 이용하여 어떠한 정책이 다른 정책에 비해 makespan이 더 좋은 경우의 개수(better), 동일한 경우(equal)의 개수, 나쁜 경우(worse)의 개수를 산출한 뒤 비교하여 직관적인 성능 비교를 한다.

5.2 입력 그래프 생성

본 논문에서 제안하는 프로세서 할당 정책을 적용하기 위한 알고리즘은 모두 분산 이기종 시스템에서 DAG의 스케줄링을 목표로 하고 있으므로 시뮬레이션을 위해서는 적절한 이기종 환경과 많은 DAG 그래프를 생성해야 한다. 프로세서 할당 정책의 공정한 성능 비교를 위해 기존의 연구에서 사용된 DAG 그래프를 인용하였으며 표준 태스크 그래프 프로젝트(STanDard task Graph Project, STDGP)[13,14]에서 제공하는 표준 DAG 그래프에 아래의 매개변수를 적용하여 DAG를 생성하고 시뮬레이션을 하였다.

STDGP에서 제공하는 표준 DAG는 노드의 계산비용과 노드 간의 통신비용에 대한 정보는 없으며 단지 그래프내의 노드의 개수와 노드들의 관계에 대한 정보만을 제공한다. 따라서 노드의 수, 통신비용, 계산비용이 적용된 DAG를 생성하기 위해 다음의 매개변수들을 입력으로 받아야만 한다.

• 통신비용 대 계산비용(Communication to Computation Ratio, CCR)

평균 통신비용에 대한 평균 계산비용의 비율이다. 만일 CCR이 매우 낮다면 이는 연산 집약적인 응용 프로그램이라 할 수 있으며, $CCR \gg 1$이면 통신 시간이 차지하는 비율이 계산 시간에 비해 상대적으로 높다는 것을 의미한다. $CCR = \{0.1, 0.5, 1.0, 2.5, 5.0\}$으로 설정하였다.

• 프로세서 이질성 척도(Range percentage value, β)

프로세서 간 속도의 이질성을 나타내는 척도이다. 큰 값을 갖는다면 어떤 노드를 실행 할 때 프로세서들 간의 계산비용에서 많은 차이를 보이게 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 균

일 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.

$$\overline{w_i}\times(1-\frac{\beta}{2}) \le w_{i,m} \le \overline{w_i}\times(1+\frac{\beta}{2}) \qquad (12)$$

본 논문에서는 $\beta = \{0.1, 0.5, 1.0, 1.5, 2.0\}$을 사용하였다.

• DAG 그래프의 노드 개수(v)

본 논문에서는 최대 750개의 노드를 이용하여 시뮬레이션 하였다. STDGP에서 제공하는 표준 DAG 중에서 노드의 개수가 50, 100, 300, 500, 750인 그래프를 인용하였다. STDGP 프로젝트에서는 노드의 수당 180개의 그래프 모양을 제공한다. 따라서 매개변수 CCR과 β값을 적용하여 총 22.5K 개의 그래프를 생성하였고 프로세서의 수는 16개로 고정하여 시뮬레이션 하였으며 다른 매개변수를 사용하였을 경우 별도로 명시하였다.

5.3 시뮬레이션 결과

여러 가지 매개변수를 이용하여 생성된 DAG 그래프를 기존의 리스트 스케줄링 알고리즘인 HCPT, HEFT, GCA, PETS에 적용하여 기존에 사용되던 프로세서 할당 정책과 본 논문에서 제안하는 프로세서 할당 정책의 성능을 비교하였다. 그림 7에서 (a),(b),(c),(d)는 16개의 프로세서에서 노드의 수 변화에 따른 평균 SLR과 Speedup을 보여준다. 각 노드의 개수마다 CCR과 β값을 변화시켜 4500개의 그래프를 생성하고 평균을 내었다. 실험에 사용된 HCPT, HEFT, GCA, PETS의 모든 알고리즘에 대하여 평균 SLR과 Speedup 모두 제안하는 프로세서 할당 정책의 성능이 기존의 프로세서 할당 정책에 비해 우위에 있는 것을 확인할 수 있다. 또한 노드의 개수가 많아질수록 성능이 향상되는 것을 볼 수 있다. 그림 7의 (a)에서 HCPT 알고리즘은 최대 13.74%의 SLR이 감소하였고 Speedup은 최대 29.25% 증가하였으며 GCA 알고리즘은 3.55%의 SLR 감소와 2.65%

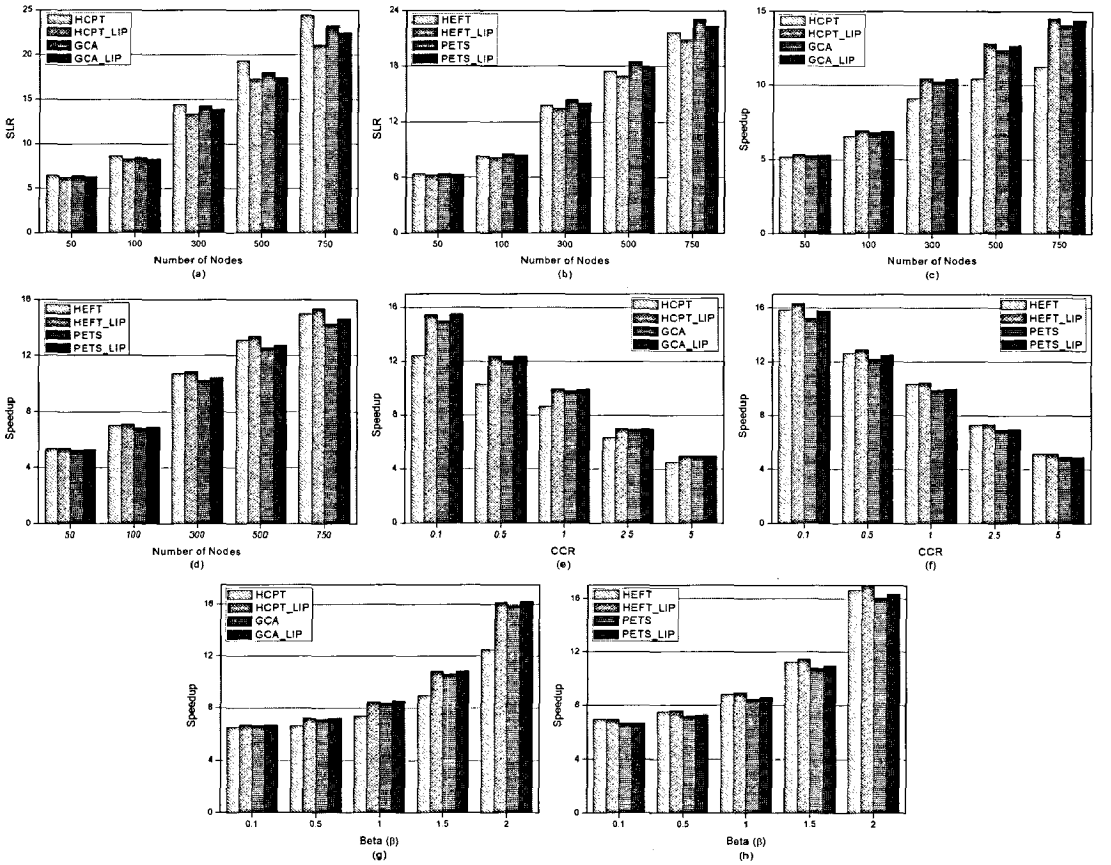

그림 7 (a) (b) 노드의 수에 따른 평균 SLR, (c) (d) 노드의 수에 따른 평균 Speedup, (e) (f) CCR에 따른 평균 Speedup, (g) (h) β (Beta)에 따른 평균 Speedup

의 Speedup이 증가 된 것을 확인할 수 있다. 그림 (b)에서 HEFT 알고리즘은 3.66%와 2.31%, PETS 알고리즘은 3.42%와 2.45%의 성능 향상을 보여주었다.

그림 7의 (e)와 (f)는 CCR에 따른 Speedup을 보여준다. CCR의 변화에 따라 HCPT는 최대 24.787%의 Speedup 증가, GCA는 3.424%의 증가, HEFT는 3.159%의 증가, PETS 알고리즘에서는 3.285%의 증가를 보여주었으며 CCR의 값이 작을수록 성능 향상이 높은 것을 확인할 수 있었다. 그림 7의 (g)와 (h)는 β 값의 변화에 따른 Speedup를 보여준다. HCPT은 29.28%의 Speedup 증가, GCA 알고리즘은 1.99%의 증가, HEFT 알고리즘에서 1.76%의 증가, PETS 알고리즘에서 1.80%의 성능 향상을 보였으며 β 값이 클수록 성능 향상이 크게 나타나는 것을 확인할 수 있었다.

표 3은 22.5K개의 그래프를 시뮬레이션 했을 때 각 알고리즘 및 프로세서 할당 정책의 차이에 대한 make-span의 길이에 대한 비교를 나타낸 것이다. 예를 들어 HCPT의 better는 HCPT_LIP보다 더 좋은 경우를 나타내고 equal은 같은 경우의 수를 나타낸다. Original과 With LIP는 모든 알고리즘에 대해 better, equal의 합을 백분율로 나타낸 것이다. 표 3에서와 같이 본 논문에서 새롭게 제안하는 프로세서 할당 정책이 기존의 프로세서 할당 정책에 비해 34.98%의 더 좋은 결과를 보이고 있다. 특히 HCPT 알고리즘과 GCA에 대해서는 큰 폭의 성능 향상이 있는 것을 볼 수 있다.

5.4 실제 응용 프로그램 그래프

알고리즘의 성능 측정을 위해 실제 사용되는 응용 프로그램인 가우스 소거법과 FFT를 DAG로 구현하여 실험 하였다.

• 가우스 소거법(Gaussian Elimination)

가우스 소거법은 선형연립방정식의 해를 구하는 데 사용하는 방법으로 전진소거와 후진대입을 이용하여 주어진 문제를 해결한다[15]. 가우스 소거법을 이용한 응용 프로그램은 연산 과정에서 병렬 소거 방법을 이용하므로 병렬 연산에 적합하여 본 논문에서 성능 측정에 사용되었다. 가우스 소거법을 사용하는 응용 프로그램은 그림 8과 같은 구성으로 되어있다. 그래프의 노드 수 v와 레벨 수 l은 행렬의 크기 m에 따라 결정된다. 행렬의 크기를 m으로 나타냈을 때, $m=6$인 경우의 그래프는 그림 9와 같다. 그래프에서 $T_{k,k}$는 피벗 칼럼(pivot column) 연산을 의미하고, $T_{k,j}$는 데이터 갱신을 의미한다. 그림 9의 임계경로는 최대 노드 개수를 갖는 $\{T_{1,1}, T_{1,2}, T_{2,2}, T_{2,3}, T_{3,3}, T_{3,4}, T_{4,4}, T_{4,5}, T_{5,5}, T_{5,6}\}$이다.

```
for k=1 to m-1 do
        T_{k,k} : { for i=k+1 to m do
                        a_ik = a_ik / a_kk }
        for j=k+1 to m do
                T_{k,j} : { for i=k+1 to m do
                                a_ij = a_ij - a_ik * a_kj }
```

그림 8 가우스 소거법 알고리즘을 나타낸 코드

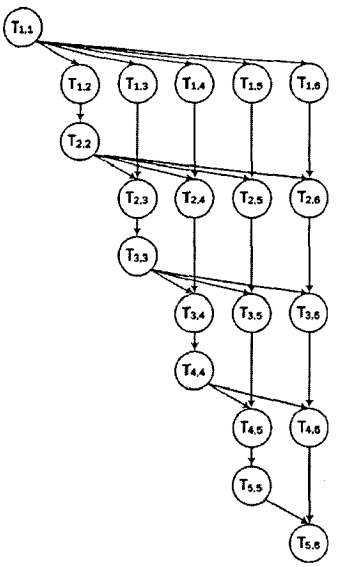

그림 9 $m=6$인 경우의 노드 그래프

5.2절에서 제시된 CCR과 β가 가우스 소거법을 사용하는 응용 프로그램의 실험에 매개변수로 사용되었다. 응용 프로그램의 그래프 구조가 알려져 있기 때문에 노드 개수, out_degree, shape parameters와 같은 다른 매개변수들의 정보는 필요 없으며 새로운 매개변수인 행렬 크기(m)가 그래프 내의 노드 개수(v) 대신 사용되었다. 가우스 소거법 그래프의 총 노드의 개수는 $(m^2+m-2)/2$이다.

• FFT(Fast Fourier Transformation)

그림 10은 8개의 데이터 포인트(data point)를 갖는 그래프의 재귀적 1차원 FFT 알고리즘이며 그림 11은 이를 나타낸 노드 그래프를 보여준다[16]. 효율적인 FFT 알고리즘의 구성에는 병렬연산이 필수적이므로 본

표 3 새로운 할당 정책을 사용했을 경우와의 성능 비교

	HCPT	HCPT_LIP	GCA	GCA_LIP	HEFT	HEFT_LIP	PETS	PETS_LIP	Original	With LIP
better	1942	16960	3899	8823	305	2680	968	3020	7.90%	34.98%
equal	3598		9778		19515		18512		57.12%	

```
FFT(W,ω)
m=length(W)
if (m=1) return (W)
F^(0)=FFT((W[0], W[2], ..., W[n-2]), ω^2)
F^(1)=FFT((W[1], W[3], ..., W[n-1]), ω^2)
for i=0 to m/2-1 do
        { F[i]=F^(0)[i]+ω^i*F^(1)[i]
          F[i+m/2]=F^(0)[i]-ω^i*F^(1)[i] }
return(F)
```

그림 10 FFT 알고리즘을 나타낸 코드

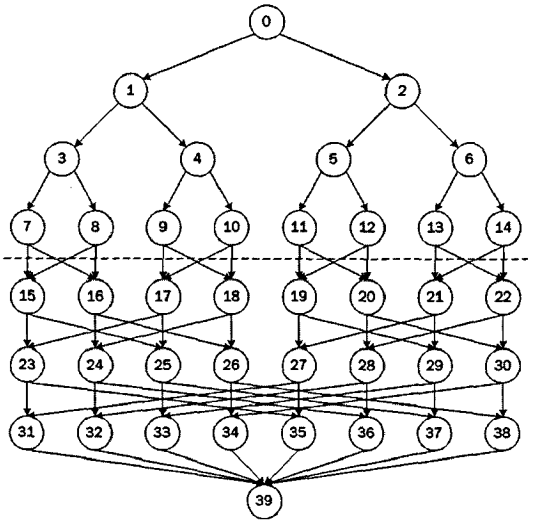

그림 11 8개의 데이터 포인트를 갖는 FFT DAG

논문의 실험에 포함하게 되었다. W는 크기가 m인 행렬로 다항식의 계수들을 포함하고 있으며, F는 알고리즘의 출력 값이 저장되는 행렬이다. 알고리즘은 재귀 호출과 버터플라이 연산(butterfly operation)의 두 단계로 구성된다. 그림 11의 노드 그래프에서 점선을 기준으로 윗부분은 노드를 재귀 호출하는 부분, 아랫부분은 버터플라이 연산을 하는 부분으로 나누어 볼 수 있다. 입력 벡터 크기가 m이라면 노드의 재귀 호출은 총 $2\times m-1$번 수행되고, 버터플라이 연산은 총 $m\times\log_2 m$번 수행된다(m은 임의의 정수 k에 대하여 $m=2^k$ 라고 가정하였다). FFT 노드 그래프에서는 같은 레벨에 있는 모든

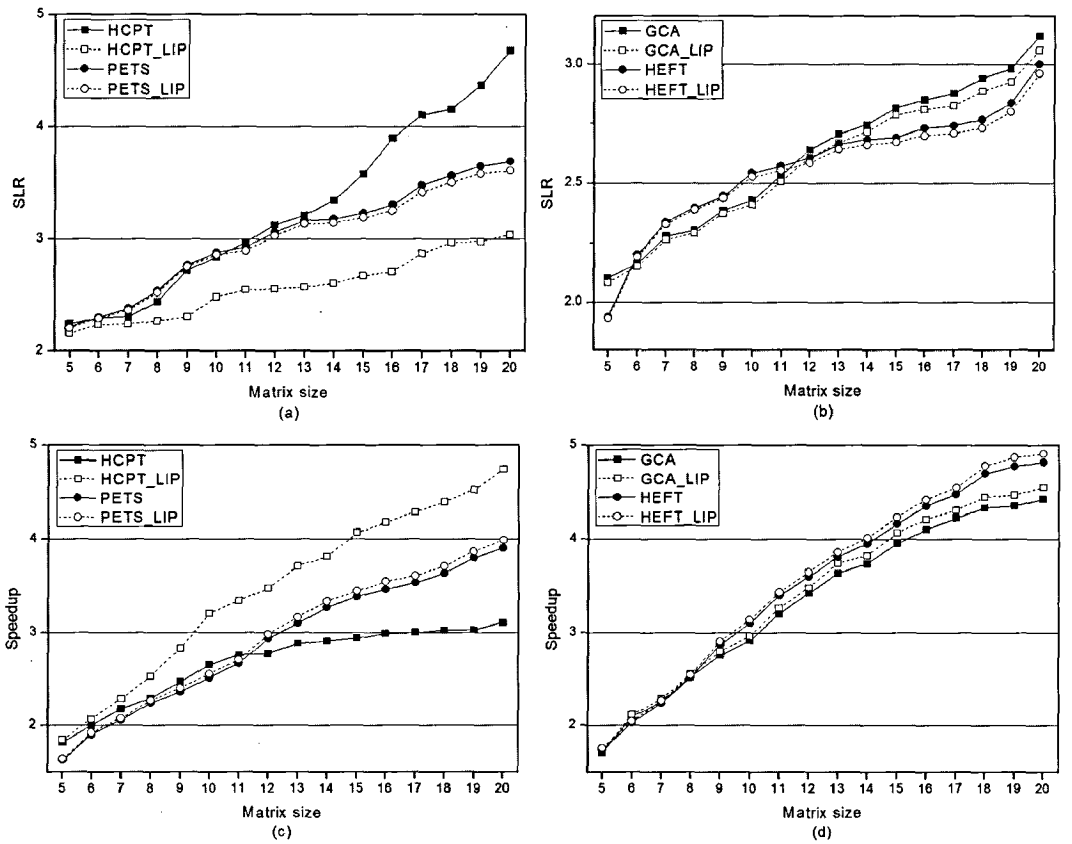

그림 12 (a) (b) 가우스 소거법 그래프에 대한 평균 SLR, (c) (d) 평균 Speedup

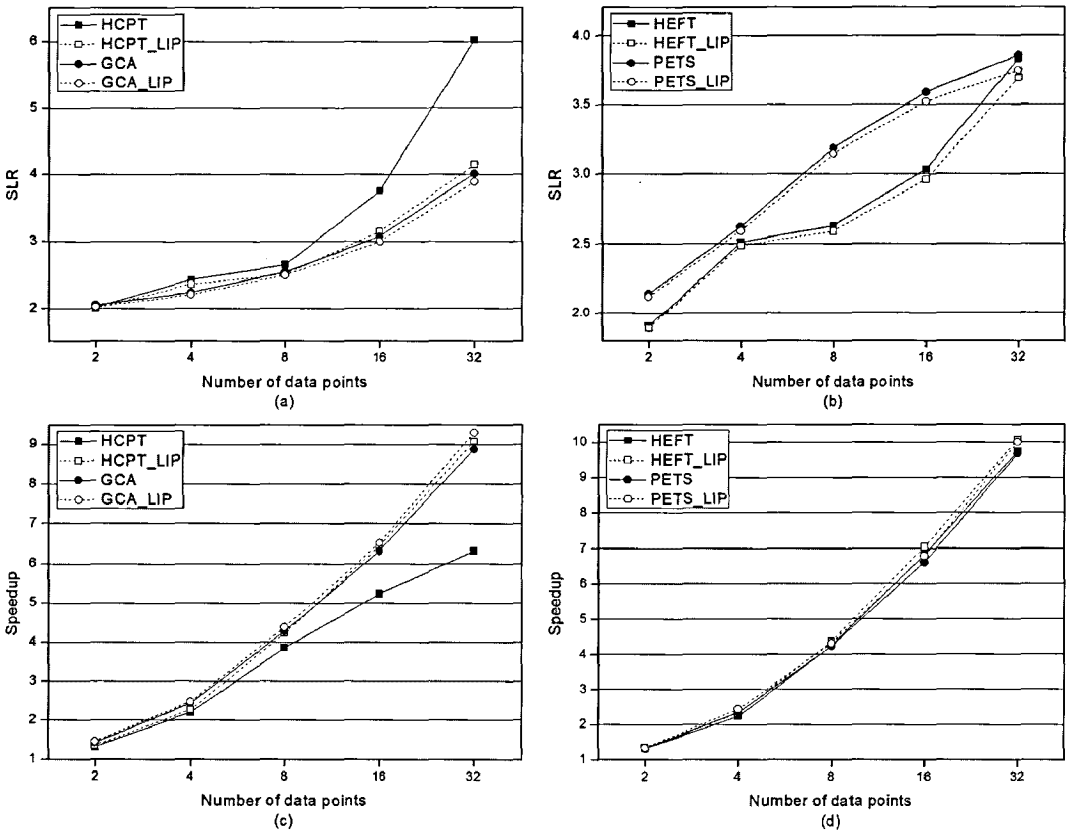

그림 13 (a) (b) FFT 그래프에 대한 평균 SLR, (c) (d) FFT 그래프에 대한 평균 Speedup

노드들의 계산비용이 동일하고 연속하는 두 레벨을 잇는 에지들의 통신비용 역시 동일하기 때문에 시작 노드로부터 종료 노드까지의 모든 경로가 임계경로이다.

FFT 실험 역시 가우스 소거법 실험과 동일하게 제시된 매개변수들 중 CCR과 β만이 실험의 매개변수로 사용되었다. 또 다른 매개변수로 FFT의 데이터 포인트가 사용되었고, 2에서 32까지 2의 거듭제곱으로 증가시켜 실험하였다.

그림 12의 (a),(b),(c),(d)는 5개의 프로세서에서 가우스 소거법에 이용되는 행렬의 크기를 5부터 20까지 하나씩 증가시켰을 때 각 알고리즘의 평균 SLR 값과 평균 Speedup 값을 보여준다. 이 실험에서 쓰인 가장 작은 그래프는 14개의 노드를, 가장 큰 그래프는 209개의 노드를 가진다. 행렬 크기를 증가시킴에 따라 더 많은 노드들이 임계경로에서 벗어나게 되고, 따라서 각 알고리즘의 전체 실행 시간(makespan)은 증가하게 된다. 그림 12의 결과 값을 통해 본 논문에서 제안하는 새로운 프로세서 할당 정책의 성능이 기존 프로세서 할당 정책의 성능보다 HCPT 알고리즘에 대하여 최대 35.12%의 SLR과 52.69%의 Speedup 향상을 보이고 있으며, GCA에 대하여 1.89%와 2.87%, HEFT에 대하여 1.30%와 1.82%, PETS에 대하여 2.28%와 2.15%의 우수한 성능을 나타내는 것을 볼 수 있다. 특히 행렬의 크기가 커질수록 LIP가 기존의 정책들에 비해 더 우수한 성능을 보이는 것을 확인할 수 있다.

그림 13은 프로세서의 개수가 8개일 때의 입력 데이터 포인트의 변화에 따른 FFT 그래프의 평균 SLR 값과 Speedup 값이다. 그림 13의 결과에서 나타나듯이 LIP를 사용하였을 때, HCPT 알고리즘에 대하여 최대 31.24%의 SLR 감소와 43.73%의 Speedup 향상이 있으며, GCA 알고리즘에 대하여 3.14%와 4.83%, HEFT 알고리즘에 대하여 3.63%와 3.43%, PETS 알고리즘에 대하여 2.88%와 3.55%의 성능 향상이 있음을 알 수 있다. 입력 데이터 포인트의 값이 커질수록 LIP가 다른 프로세서 할당 정책들에 비해 더 좋은 결과 값을 나타내고 있다.

6. 결 론

본 논문에서는 기존 리스트 스케줄링 알고리즘의 노

드 우선순위 결정 방법을 이용하여 부모-자식노드의 관계에 따라 프로세서를 할당하게 되는 새로운 프로세서 할당 정책인 LIP를 제안하였다. 기존의 연구에서는 부모-자식간의 관계를 고려하지 않고 최소 완료 시간만을 이용한 삽입 혹은 비 삽입 기반 할당 정책을 이용하여 노드를 프로세서에 할당하였으나 본 논문에서는 노드의 부모-자식 관계를 고려하여 노드를 할당함으로써 불필요한 통신비용의 발생을 없애는 방법으로 성능의 향상을 이룰 수 있었다. 공정한 시뮬레이션을 위해 기존의 문헌에서 사용한 DAG 그래프 및 실제 응용 프로그램의 노드 그래프를 인용하여 실험하였으며 또한 기존의 실험에서 사용된 노드의 개수보다 많은 노드의 그래프를 사용하여 대용량의 병렬 프로그램 처리를 고려하였다. 22.5K개의 DAG와 실제 응용 프로그램의 노드 그래프를 이용하여 실험한 결과 기존에 사용되던 프로세서 할당 정책보다 좋은 성능을 보였으며 특히 CCR 값이 작고 β 값이 클수록 LIP 정책의 성능 향상이 높은 것을 확인할 수 있었다.

참 고 문 헌

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

[2] 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.

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

[4] D.Feitelson, L. Rudolph, U. Schwiegelshohm, K. Sevcik, P. Wong, "Theory and practice in parallel job scheduling," *JSSPP*, pp.1-34, 1997.

[5] Y. Kwok, I. Ahmed, "Benchmarking the task graph scheduling a algorithm," *Proc. IPPS/SPDP*, 1998.

[6] 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.

[7] W. Yoon, J. Yoon, C. Lee, H. Gim, S. Choi, "An Efficient List Scheduling Algorithm in Distributed Heterogeneous Computing System," *Journal of IEEK : CI*, vol.46, no.3, May. 2009. (in Korea)

[8] 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.

[9] Ching-Hsien Hsu, Chih-Wei Hsieh and Chao-Tung Yang, "A Generalized Critical Task Anticipation Technique for DAG Scheduling," *ICA3PP, LNCS 4494*, pp.493-505, 2007.

[10] Michael A. Iverson, F. Ozgunner and Gregory J. Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," *Proceeding Heterogeneous Computing Workshop*, pp.93-100, 1995.

[11] E. Ilavarasan and P. Thambidurai, "Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments," *Journal of Computer Sciences*, 3(2), pp.94-103, 2007.

[12] E. Ilavarasan, P. Thambidurai and R. Mahilmannan, "High Performance Task Scheduling Algorithm for Heterogeneous Computing System," *LNCS* 3718, pp.193-203, 2005.

[13] Takao Tobita and Hironory kasahara, "A Standdard Task Graph Set for Fair Evaluation of Multiprocessor Scheduling Algorithms," *Journal of Scheduling*, 5, pp.379-394, 2002.

[14] http://www.kasahara.elec.waseda.ac.jp.

[15] M. Cosnard, M. Marrakchi, Y. Robert, and D. Trystram, "Parallel Gaussian Elimination on an MIMD Computer," *Parallel Computing*, vol.6, pp.275-295, 1988.

[16] 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.

윤 완 오

2000년 경기대학교 전자공학과(학사). 2002년 인하대학교 전자공학과(석사). 2002년~현재 인하대학교 전자공학과 박사과정. 관심분야는 분산 처리 시스템, 임베디드 시스템, 컴퓨터 구조, 컴퓨터 네트워크 등

송 인 성

2009년 인하대학교 전자공학과(학사). 2009년~현재 인하대학교 전자공학과 석사과정. 관심분야는 컴퓨터 아키텍처, 분산 처리 시스템, 병렬 프로그래밍

윤 준 철

2006년 한국산업기술대 전자 공학(학사) 2008년 2월 인하대학교 전자공학과(석사). 2009년~현재 인하대학교 전자공학과 박사 과정. 관심분야는 병렬 및 분산 처리 시스템, 컴퓨터구조

최 상 방

1981년 한양대학교 전자공학과(학사). 1981년~1986년 LG 정보통신(주). 1988년 University of washinton(석사). 1990년 University of washington(박사). 1991년~현재 인하대학교 전자공학과 교수 관심분야는 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, Fault-tolerant computing