논문 2010-47CI-1-15

분산 이기종 컴퓨팅 시스템을 위한 새로운 고성능 리스트 스케줄링 알고리즘

(A Novel High Performance List Scheduling Algorithm for Distributed Heterogeneous Computing Systems)

윤 완 오*, 윤 준 철*, 윤 정 희*, 최 상 방**

(Wan-Oh Yoon, Jun-Chul Yoon, Jung-Hee Yoon, and Sang-Bang Choi)

요 약

분산 이기종 컴퓨팅 시스템(Distributed Heterogeneous Computing System, DHCS)에서 방향성 비순환 그래프(Directed Acyclic Graph, DAG)의 효율적인 스케줄링은 시스템의 높은 성능을 만드는데 매우 중요한 역할을 한다. 본 논문은 DHCS에서 고성능의 새로운 스케줄링 알고리즘인 LCFT(Levelized Critical First Task)을 제안한다. LCFT 알고리즘은 DHCS에서 스케줄링을 위해 효율적인 태스크 선택 방법을 이용하는 리스트 스케줄링 기반의 알고리즘이다. LCFT 알고리즘의 복잡도는 $O(v+e)(p+\log v)$을 갖는다. LCFT의 성능 비교를 위해 다양한 DAG 그래프를 이용하여 기존의 알고리즘인 PETS, HPS, HCPT, GCA와 스케줄링의 길이와 속도를 실험하였으며 실험 결과 LCFT 알고리즘이 다른 알고리즘 보다 성능 향상이 있는 것을 확인할 수 있었다.

Abstract

Efficient Directed Acyclic Graph(DAG) scheduling is critical for achieving high performance in Distributed Heterogeneous computing System(DHCS). In this paper, we present a new high-performance scheduling algorithm, called the LCFT(Levelized Critical First Task) algorithm, for DHCS. The LCFT algorithm is a list-based scheduling that uses a new attribute to efficiently select tasks for scheduling in DHCS. The complexity of LCFT is $O(v+e)(p+\log v)$. The performance of the algorithm has been observed by its application to some practical DAGs, and by comparing it with other existing scheduling algorithms such as PETS, HPS, HCPT and GCA in terms of the schedule length and SpeedUp. The comparison studies show that LCFT significantly outperforms PETS, HPS, HCPT and GCA in schedule length, SpeedUp.

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

I. 서 론

네트워크 기술의 발달로 서로 다른 성능을 갖는 프로세서들 간에 효율적인 통신이 가능하게 되면서 병렬 프로그램을 동시에 처리할 수 있는 환경이 만들어 졌는데 이를 분산 이기종 컴퓨팅 시스템(Distributed Heterogeneous Computing System, DHCS)이라 한다. DHCS에서 복잡한 연산을 필요로 하는 어플리케이션을 효율적으로 실행하기 위해서는 두 가지 중요한 문제가 있는데 먼저 큰 규모의 어플리케이션을 DHCS 환경의 병렬성을 최대한 활용하기 위해 작은 태스크로 이루어진 병렬 프로그램으로 만들어야 한다. 두 번째는 병렬 프로그램을 구성하는 태스크들의 실행시간이 최적이 되도록 DHCS를 구성하는 프로세서에 할당해야 한다. 일반적으로 병렬 프로그램은 선행제약이 없는 독립된 태스크들로 구성된 프로그램과 방향성 비순환 그래프(Directed Acyclic Graph, DAG)로 표현 할 수 있는 프

* 학생회원, ** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering Inha University)
접수일자 : 2009년7월23일, 수정완료일 : 2010년1월8일

로그램 그리고 주기를 가지며 방향성이 없는 그래프로 표현할 수 있는 프로그램으로 나눌 수 있다. 본 논문에서는 DAG로 표현 할 수 있는 병렬 프로그램을 입력 그래프로 사용한다. DHCS에서 DAG로 표현 할 수 있는 병렬 프로그램의 실행 성능은 태스크를 프로세서에 할당하는 스케줄링 알고리즘에 많이 의존한다. 스케줄링의 목적은 DAG를 이루는 태스크의 선행제약을 만족하면서 각 프로세서에 태스크를 할당하여 병렬 프로그램의 전체 실행 시간(makespan)을 최소화하는 것이다. 최적의 스케줄링은 전체 실행 시간에 성능 향상을 가져다주지만 다항 시간(polynomial time)내에 결과를 얻을 수 없는 NP-complete이다[1~3]. 그래서 많은 연구들이 만족할만한 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 있다. 정적 스케줄링은 태스크들 사이의 통신비용이나 각 태스크의 계산비용과 같이 스케줄링에 필요한 정보를 미리 알고 병렬 프로그램이 실행되기 전 컴파일(compile) 시간에 태스크 스케줄링을 실행한다. 이와는 반대로 동적 스케줄링은 병렬 프로그램이 실행되는 동안에 스케줄링에 필요한 정보를 획득하여 스케줄링 하는 방법이다. 정적 스케줄링은 다시 클러스터 스케줄링, 태스크 복제 스케줄링, 리스트 스케줄링으로 나눌 수 있다. 클러스터 스케줄링은 태스크들 사이의 통신비용을 고려하여 여러 개의 태스크를 하나의 클러스터로 묶어 최적의 프로세서에 할당함으로서 태스크 사이의 통신 시간을 줄여 최적의 결과를 얻기 위한 알고리즘이다[4~5]. 태스크 복제 스케줄링은 태스크들 간의 상호 의존성에 의한 통신 시간을 감소시키기 위해 선행 태스크의 복제를 통해 스케줄링을 수행하는 알고리즘으로 다른 스케줄링에 비해 복잡도가 커지는 단점은 있으나 스케줄링의 결과는 다른 알고리즘에 비해 향상된 결과를 보인다[6~9]. 리스트 스케줄링은 태스크 간의 상호 제약을 충족시키면서 몇몇 우선순위 함수에 기초하여 태스크의 우선순위를 결정하고 결정된 우선순위 순서대로 최적의 프로세서에 할당하는 방법으로 대표적인 알고리즘으로는 HCPT(Heterogeneous Critical Parent Trees)[10], HRPS(Heterogeneous Rank-Path Scheduling)[11], HEFT(Heterogeneous Earliest Finish Time)[12], CPOP(Critical path on a processors)[12], MH (Mapping Heuristic)[13], GCA(Generalized Critical-task Anticipation)[14], LMT(Levelized Min Time)[15], PETS(Performance Effective Task Scheduling Algorithm)[16], HPS(High Performance- task Scheduling)[17]와 같은 알고리즘이 있다. 이와 같은 리스트 스케줄링 알고리즘 중에서도 LMT, PETS, HPS 알고리즘은 레벨화된 리스트 스케줄링 알고리즘이다. 레벨화된 리스트 스케줄링 알고리즘은 태스크의 우선순위를 결정하기 전에 태스크들을 선행제약이 없는 태스크들끼리 그룹화 하는 알고리즘을 의미한다. 독립적인 태스크들끼리 그룹화를 함으로써 태스크의 우선순위를 결정할 때 사용하는 우선순위 함수에 여러 매개 변수를 추가 하더라도 태스크들 사이에 선행제약을 지키면서 우선순위를 결정할 수 있다. 리스트 스케줄링은 비교적 다른 알고리즘에 비해 낮은 복잡도를 갖는다.

본 논문에서는 레벨화된 리스트 스케줄링의 한 방법으로서 새로운 정적 스케줄링 알고리즘인 LCFT (Levelized Critical First Task)을 제안한다. LCFT는 태스크의 우선순위를 결정할 때 태스크의 실행비용과 부모 태스크로부터 데이터를 전달받기 위한 통신비용 그리고 태스크부터 마지막 태스크까지의 임계경로(critical path)를 이용하여 계산한다. 기존의 알고리즘과는 다른 새로운 우선순위 함수를 이용함으로서 낮은 복잡도와 높은 성능을 가진다. 제안된 알고리즘의 성능을 확인하기 위해 여러 가지 매개변수에 의해 만들어진 DAG를 활용하여 기존의 대표적인 레벨화된 리스트 스케줄링 알고리즘인 PETS, HPS와 리스트 스케줄링 알고림즘인 HCPT, GCA와 성능 비교를 하였다. 비교 결과 본 논문에서 제안한 LCFT 알고리즘의 성능이 다른 알고리즘의 성능에 비해 향상된 것을 확인 할 수 있었다.

본 논문은 다음과 같이 이루어져 있다. Ⅱ장에서는 본 논문에서 제안한 알고리즘이 적용되는 환경과 알고리즘을 설명하기 위해 사용되는 용어를 설명하고 Ⅲ장에서는 본 논문에서 비교한 기존의 대표적인 리스트 스케줄링 알고리즘인 PETS, HPS, HCPT, GCA을 소개한다. Ⅳ장에서는 본 논문에서 제안한 알고리즘을 설명하고 Ⅴ장에서는 실제 사용되는 어플리케이션의 그래프와 임의의 그래프를 기반으로 실험을 통해 제안된 알고리즘의 성능 분석을 기술한다. 마지막으로 Ⅵ장에서 결론을 맺는다.

Ⅱ. 스케줄링을 위한 문제 정의

DHCS에서 정적 스케줄링을 수행하기 위한 병렬 프로그램은 그림 1과 같이 방향성 비순환 그래프(DAG) $G=(T,E)$로 표현 할 수 있다. 여기서 T는 n개의 태스크 t의 집합이고 태스크들 중에서 i번째 태스크는 t_i로 표현한다. E는 통신 에지(communication edge) e의 집합이며 태스크 t_i에서 t_j로의 방향성을 갖는 에지 $e(t_i,t_j)$가 존재한다면 t_i는 t_j의 부모태스크(parent task)라고 부르고 $t_i \in pred(t_j)$로 표현되며 t_j는 t_i의 자식태스크(child task)라고 부르고 $t_j \in succ(t_i)$로 표현된다. 그림 1의 태스크 1처럼 부모태스크가 존재하지 않는 태스크를 시작태스크(entry task) t_s라 하고 태스크 10처럼 자식태스크가 없는 태스크를 출력태스크(exit task) t_e라 정의한다. 본 논문에서 제안한 스케줄링을 하기 위해서는 오직 한 개의 입력태스크와 출력태스크가 필요하다. 만약 2개 이상의 입력태스크와 출력태스크가 있다면 이 태스크들을 묶을 수 있는 의사 태스크(pseudo task)를 추가 할 수 있으며 의사 태스크의 계산비용과 통신비용은 모두 0값을 가지고 이 태스크는 스케줄링 상에 아무런 영향을 주지 않는다고 가정한다.

DHCS는 m개의 서로 다른 성능을 갖는 프로세서의 집합 P로 표현되고 프로세서들은 서로 완전연결(fully connected)되어 있다고 가정한다. W는 표 1과 같이 태스크 t_i가 프로세서 $p_k \in P$에서 실행될 때의 계산비용(computation cost) $w_{i,k} \in W$의 요소로 이루어진 $n \times m$ 크기의 계산비용 행렬이다. 스케줄링 알고리즘을 수행하기 전에 각 태스크의 평균 계산비용을 계산해야 하는데 예를 들어 m개의 프로세서에서 태스크 t_i의 평균 계산비용 $t_i^{\bar{w}}$는 식(1)과 같다.

$$t_i^{\bar{w}} = \sum_{k=1}^{m} \frac{w_{i,k}}{m} \tag{1}$$

두 개의 프로세서 사이에 전송되는 데이터의 바이트당 전송비용은 $m \times m$ 크기를 가지는 행렬 R로 표현되고 각 프로세서의 통신 시작비용(startup cost)은 1차원 벡터 S로 주어진다. 프로세서 p_k에 스케줄링 된 태스크 t_i로부터 프로세서 p_l에 스케줄링 된 태스크 t_j까지 μ바이트 크기의 데이터를 전송할 때 그에 해당하는 에지 $e(t_i,t_j) \in E$의 통신비용 $c_{(i,k)(j,l)}$은 식 (2)와 같이 정의된다[10~11].

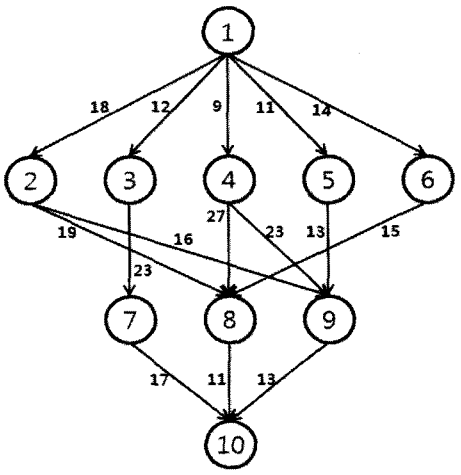

그림 1. 10개의 태스크를 가지는 DAG의 예
Fig. 1. A example of DAG with 10 tasks.

표 1. 계산비용 행렬(W)과 평균 계산비용($t_i^{\bar{w}}$)
Table 1. Computation Cost matrix(W) and average computation cost($t_i^{\bar{w}}$).

$task$	p_1	p_2	p_3	$t_i^{\bar{w}}$
t_s	14	16	9	13
t_2	13	19	18	16.67
t_3	11	13	19	14.33
t_4	13	8	17	12.67
t_5	12	13	10	11.67
t_6	13	16	9	12.67
t_7	7	15	11	11
t_8	5	11	14	10
t_9	18	12	20	16.67
t_e	21	7	16	14.67

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

여기서 S_k는 프로세서 p_k의 통신 시작시간(sec), $\mu_{i,j}$는 태스크 t_i부터 태스크 t_j로 전달되는 데이터 전송 양(bytes) 그리고 $R_{k,l}$은 p_k으로부터 p_l까지의 바이트 당 통신비용(sec/byte)이다. 프로세서 p_k와 p_l이 같은 프로세서일 경우 통신비용은 0값을 가지며 서로 다른 프로세서라면 통신비용은 통신 시작시간과 μ바이트 크기의 데이터가 전달되는데 필요한 시간의 합으로 나타낼 수 있다. 그림 1의 입력 그래프 DAG에서 에지 $e(t_i,t_j) \in E$에 표현된 수치는 각 프로세서의 통신 시작시간 S와 프로세서 사이의 바이트당 전송비용 R, 데이터 전송 양 μ의 평균을 이용하여 계산된 평균 통신비용 $c_{i,j}$을 표현한 것이다.

Ⅲ. 관련 연구

기존 논문의 실험에서 HEFT 알고리즘이 CPOP, MH, LMT 보다 성능이 높다는 것을 보여주었고 PETS, HPS, GCA 알고리즘은 HEFT, CPOP, LMT 보다 성능이 높다는 것을 보여 주었다. 따라서 본 논문에서 제안한 LCFT 알고리즘의 성능을 증명하기 위해 레벨화된 리스트 스케줄링 알고리즘인 PETS, HPS와 리스트 스케줄링 알고리즘인 HCPT, GCA와 성능 비교를 하였다. 이 장에서는 PETS, HCPT, HPS, GCA 알고리즘에 대해 설명한다.

1. PETS(Performance Effective Task Scheduling Algorithm)

PETS 알고리즘은 레벨화된 리스트 스케줄링 알고리즘으로 세 단계로 이루어져 있다. 첫 번째 단계인 레벨 정렬 단계에서는 주어진 DAG 그래프의 각 태스크를 독립적인 태스크들끼리 그룹화하여 레벨화를 한다. 두 번째 단계인 태스크 우선순위 결정 단계에서는 레벨 1부터 시작하여 각 레벨의 태스크에 우선순위 함수를 적용하여 태스크의 우선순위를 결정한다. 우선순위 함수는 태스크의 실행시간, 자식태스크의 통신비용의 합과 부모태스크 중에서 가장 큰 우선순위 값의 합으로 구성되며 우선순위 함수에 의해 구해진 값을 이용하여 태스크를 내림차순으로 정렬한다. 마지막 단계인 프로세서 선택 단계에서는 두 번째 단계에서 결정된 태스크의 순서에 따라 삽입정책(insertion-based)을 이용하여 최소의 실행 완료 시간을 가지는 프로세서에 할당한다. 알고리즘의 복잡도는 $O(v+e)(p+\log v)$의 크기를 갖는다. 여기서 v는 태스크의 개수를 나타내고, e는 에지의 개수 그리고 p는 프로세서의 개수를 나타낸다.

2. HCPT(Heterogeneous Critical Parent Trees)

HCPT 알고리즘은 두 단계로 이루어져 있다. 첫 번째 단계인 태스크 우선순위 결정단계에서는 가장 빠른 평균 시작시간과 가장 늦은 평균 시작시간을 구하고 두 값의 차가 0인 태스크를 임계경로의 태스크로 결정하고 임계경로를 이루는 태스크들을 스택에 넣고 비어있는 큐를 만든다. 다음으로 스택의 최상위에 있는 태스크의 부모태스크가 큐에 있는지 확인한다. 큐에 부모태스크가 없다면 부모태스크를 스택에 *push*하고 큐에 부모태스크가 있다면 스택의 최상위 태스크를 *pop*하여 큐에 넣고 스택이 빌 때까지 이와 같은 동작을 반복한 후 큐에 들어있는 순서대로 태스크의 우선순위가 결정된다. 두 번째 단계에서는 큐에 있는 태스크를 하나씩 가장 빠른 시간에 끝낼 수 있는 프로세서에 할당하게 된다. 알고리즘의 복잡도는 $O(v^2 \times p)$이다. 이 알고리즘의 문제점은 부모태스크가 많을 경우 어떤 부모태스크를 먼저 선택 하여 스택에 *push* 할 것 인지에 대한 정의가 없다. 임의대로 부모태스크를 선택할 경우 태스크의 우선순위가 변화 할 수 있으며 그로 인한 성능 향상을 기대할 수 없다.

3. HPS(High Performance-task Scheduling)

HPS 알고리즘도 PETS와 마찬가지로 레벨화된 알고리즘으로 레벨 정렬 및 프로세서 선택 단계는 PETS와 동일하다. 두 번째 단계인 태스크 우선순위 결정 단계에서는 자식태스크의 통신비용 중 가장 큰 값, 부모태스크의 통신비용 중 가장 큰 값 그리고 부모태스크의 우선순위 값 중에서 가장 큰 값의 합을 이용하여 우선순위를 결정한다. 즉, 태스크의 실행비용은 고려하지 않고 통신비용만을 고려한 알고리즘으로 복잡도는 $O(v+e)(p+\log v)$이다.

4. GCA(Generalized Critical-task Anticipation)

GCA 알고리즘은 두 단계로 이루어져 있으며 두 번째 단계인 프로세서 선택 단계는 PETS, HPS 알고리즘과 마찬가지로 삽입정책을 이용하여 최소의 실행 시간을 갖는 프로세서에 태스크를 할당한다. 첫 번째 단계인 태스크 우선순위 결정 단계에서 우선 각 태스크의 우선순위 값을 마지막 태스크부터 태스크의 평균 실행시간과 자식태스크 중에서 가장 큰 통신비용과 우선순위 값을 이용하여 재귀적으로 계산한다. 이 알고리즘이 기존의 알고리즘인 HEFT 알고리즘과 다른 점은 우선순위 값을 내림차순으로 정렬하여 태스크의 우선순위를 결정하지 않고 우선순위 값이 높은 태스크에 높은 우선순위를 부여하면서 자식 태스크에 주는 영향에 따라 순위를 결정하는 방법을 사용한다. 알고리즘의 복잡도는 $O(v^2 \times p)$이다.

Ⅳ. 제안된 스케줄링 알고리즘

본 논문에서는 태스크의 실행 시간과 현재 태스크부터 마지막 태스크까지의 임계경로 그리고 부모태스크

로부터 데이터를 받는데 필요한 통신비용을 고려한 새로운 알고리즘인 LCFT를 제안한다. LCFT는 3단계로 구성되는데 선행제약이 있는 DAG 그래프의 태스크들을 독립적인 태스크의 그룹으로 만들어주는 레벨 정렬 단계와 레벨로 분리된 태스크의 우선순위를 결정하는 우선순위 결정 단계 그리고 결정된 우선순위로 태스크를 프로세서에 할당하는 프로세서 선택 단계로 이루어져 있다. 이 장에서는 알고리즘의 각 단계에 대해 기술한다.

1. 레벨 정렬 단계

본 논문에서는 입력 그래프 DAG를 이루는 각 태스크의 우선순위를 결정하기 전에 너비 우선 검색(Breadth First Search, BFS)을 이용하여 각 태스크를 독립적인 태스크의 집합으로 레벨화 한다. 즉 같은 레벨에 속한 태스크들 사이에는 선행제약이 없으며 독립적으로 실행이 가능하게 된다. 그림 1에서 시작 태스크 t_s을 레벨 1(L_1)에 할당하고 다음 태스크 t_1이 L_1에 있는 t_s태스크와 선행제약이 있으므로 1 만큼 증가된 새로운 레벨 2(L_2)을 만들고 t_1을 할당한다. 마찬가지로 다음 태스크 t_2는 L_2의 t_1과 선행제약이 없으므로 L_2에 할당된다. 이 과정을 마지막 태스크 t_e가 할당 될 때까지 반복한다. 결과적으로 그림 1의 각 태스크는 $L_1=\{t_s\}$, $L_2=\{t_2,t_3,t_4,t_5,t_6\}$, $L_3=\{t_7,t_8,t_9\}$, $L_4=\{t_e\}$ 레벨로 할당된다. 태스크들을 레벨로 나누는 이유는 서로 상호 제약이 없는 태스크들로 그룹화 함으로서 태스크의 우선순위를 결정하는데 부모태스크로부터의 데이터를 전달받는 통신비용을 고려하기 위해서이다.

2. 우선순위 결정 단계

두 번째 단계는 각 레벨에 있는 태스크의 우선순위를 결정하는 단계이다. 본 논문에서는 태스크의 우선순위를 결정하기 위해 3가지 요소를 고려하였다. 첫 번째 요소는 계산비용이 높은 태스크에게 우선권을 주는 것이다. 태스크 t_i의 평균 계산비용 $t_i^{\overline{w}}$는 식 (1)을 이용하여 계산한다. 태스크를 실행하기 위해서는 부모태스크로부터 데이터를 모두 전달받아야 한다. 따라서 태스크의 우선순위를 고려함에 있어 두 번째 요소는 부모 태스크로부터 가장 늦게 데이터를 전달받는 태스크의 우선순위를 높게 하는 것이다. 그러기 위해 태스크 t_i가 n개의 부모태스크로부터 데이터를 전달받은데 걸리는 평균 통신비용 $ADRC(t_i)$(Average Data Receive Cost)를 식 (3)을 이용하여 계산한다.

표 2. 그림 1의 각 태스크에 $t_i^{\overline{w}}$, $ADRC(t_i)$, $CCT(t_i)$ $rank(t_i)$ 그리고 우선순위 값.

Table 2. The $t_i^{\overline{w}}$, $ADRC(t_i)$, $CCT(t_i)$ $rank(t_i)$ and priority values for the tasks in Fig1.

Level	Task	$t_i^{\overline{w}}$	$ADRC(t_i)$	$CCT(t_i)$	$rank(t_i)$	Priority
1	t_s	13	0	97.01	110.01	1
2	t_2	16.67	18	62.34	97.01	1
	t_3	14.33	12	62.34	88.67	2
	t_4	12.67	9	62.34	84.01	5
	t_5	11.67	11	62.34	85.01	4
	t_6	12.67	14	58.67	85.34	3
3	t_7	11	23	28.34	62.34	2
	t_8	10	20.33	28.34	58.67	3
	t_9	16.67	17.33	28.34	62.34	1
4	t_e	14.67	13.67	0	28.34	1

$$ADRC(t_i)=\sum_{j=1}^{n} c_{j,i}/n \tag{3}$$

여기서 $c_{j,i}$는 태스크 t_i의 j번째 부모태스크와의 통신비용이다. 따라서 $ADRC(t_i)$는 태스크 t_i가 부모태스크로부터 데이터를 전달 받기 위한 통신비용의 평균이다. 마지막으로 가장 비용이 높은 자식 태스크를 가지고 있는 태스크에게 우선순위를 높게 하기 위해 $CCT(t_i)$(Critical Child Task)을 식 (4)을 이용하여 계산한다.

$$CCT(t_i)=Max\{rank(t_1),rank(t_2),...,rank(t_j)\}$$
$$where,\ \{t_1,t_2,...,t_j\}\in succ(t_i),\ CCT(t_e)=0 \tag{4}$$

여기서 $succ(t_i)$는 t_i의 자식태스크의 집합을 의미하며 마지막 태스크 t_e는 자식태스크가 없기 때문에 $CCT(t_e)$=0의 값을 갖게 된다. 태스크 t_i의 우선순위를 결정하기 위한 우선순위 함수 $rank(t_i)$는 3개의 인자를 가진 식 (5)와 같이 정의 된다.

```
//Level Sorting Phase
1. Level Sort the DAG(T,E)
2. k=|Level|  //number of level
//Task Prioritizing Phase
3. While (k != 0)
4.      For each tasks t_i at each level L_k do
5.          Compute t_i^w, ADRC(t_i) and CCT(t_i);
6.          Compute rank(t_i) = t_i^w + ADRC(t_i) + CCT(t_i);
7.          Construct a priority queue(Q) using rank(t_i);
8.      End For
9.        k=k-1;
10. End While
//Processor Selection Phase
11. While there are unscheduling tasks in the Q
12.     Select the highest priority task, t_i from the Q for scheduling
13.     For each processor p_k in the processor set P do
14.         Compute ect(t_i , p_k) value using the insertion-based scheduling policy
15.     End for
16.     Assign the task t_i to the processor p_k which minimizes the ect of task t_i
17. End While
```

그림 2. 제안된 LCFT 알고리즘
Fig. 2. Proposed LCFT algorithm.

$$rank(t_i) = t_i^{\overline{w}} + ADRC(t_i) + CCT(t_i) \quad (5)$$

태스크의 우선순위 계산은 $rank(t_i)$ 함수를 이용하여 마지막 레벨부터 재귀적으로 계산되고 각 레벨안에서 태스크의 우선순위는 $rank(t_i)$값이 큰 태스크가 우선순위가 높게 된다. 예를 들어 그림 1, 표 1과 같이 입력그래프와 계산비용이 주어졌을 경우 레벨 L_4에 있는 마지막 태스크 t_e는 자식태스크가 없으므로 $CCT(t_e)$는 0의 값을 가지게 되고 평균 계산비용 $t_e^{\overline{w}}$는 14.67의 값을 가진다. 그리고 $ADRC(t_e)$는 ((17+11+13)/3) = 13.67 의 값을 가지게 되어 우선순위 함수 $rank(t_e)$의 값은 (14.67+13.67+0) = 28.34를 가지게 된다. 이처럼 레벨 L_4부터 레벨 L_1에 포함된 각 태스크의 $t_i^{\overline{w}}$, $ADRC(t_i)$, $CCT(t_i)$, $rank(t_i)$ 그리고 우선순위 값을 계산하여 표 2에 나타내었다. 만약 하나의 레벨에 $rank(t_i)$값이 같은 태스크가 있을 경우 본 논문에서는 실행비용이 큰 값에 우선순위를 높게 적용한다. 예를 들어 표 2에서 태스크 7과 9는 같은 $rank(t_i)$값을 가지지만 태스크 9가 7보다 실행비용이 높기 때문에 우선순위가 높게 된다. 최종적으로 태스크의 우선순위는 $\{t_s, t_2, t_3, t_6, t_5, t_4, t_9, t_7, t_8, t_e\}$의 값을 가진다.

3. 프로세서 선택 단계

태스크 우선순위 결정 단계에서 각 태스크의 우선순위를 결정하고 나면, 우선순위가 가장 높은 태스크부터 적절한 프로세서를 선택하여 할당하게 된다. 최적의 프로세서에 태스크를 할당하기 위해서 우선 초기 시작 시간 est(earliest start time)와 태스크의 실행 완료 시간 ect(earliest completion time)를 정의한다. est는 태스크 t_i가 프로세서 p_k에서 실행 가능한 시간을 의미하며 식 (6)과 같이 정의한다.

$$est(t_i, p_k) = \begin{cases} 0 & \text{if } t_i = t_s \\ \max\limits_{t_j \in pred(t_i)} \{Avail[p_k], DAT(t_j)\} & otherwise \end{cases} \quad (6)$$

여기서 $pred(t_i)$는 태스크 t_i의 부모태스크의 집합을 의미하며 $Avail[p_k]$은 프로세서 p_k에서 태스크를 실행할 수 있는 실행 가능 시간을 의미한다. 처음 시작 태스크가 프로세서 p_k에서 실행 될 경우 $est(t_s, p_k) = 0$의 값을 가지게 된다. $DAT(t_j)$는 $\max(ect(t_j) + k \cdot c_{j,i})$로 정의할 수 있으며 t_i의 부모태스크 중에서 가장 늦게 데이터를 전달하는 태스크를 의미한다. 여기서 상수 k는 부모태스크가 같은 프로세서에 할당 되었을 경우 0의 값을 가지고 그 외에는 1의 값을 가진다. 태스크 실행 완료 시간 ect는 태스크의 시작 시간에 실행시간을 더한 값으로 식 (7)과 같이 정의된다.

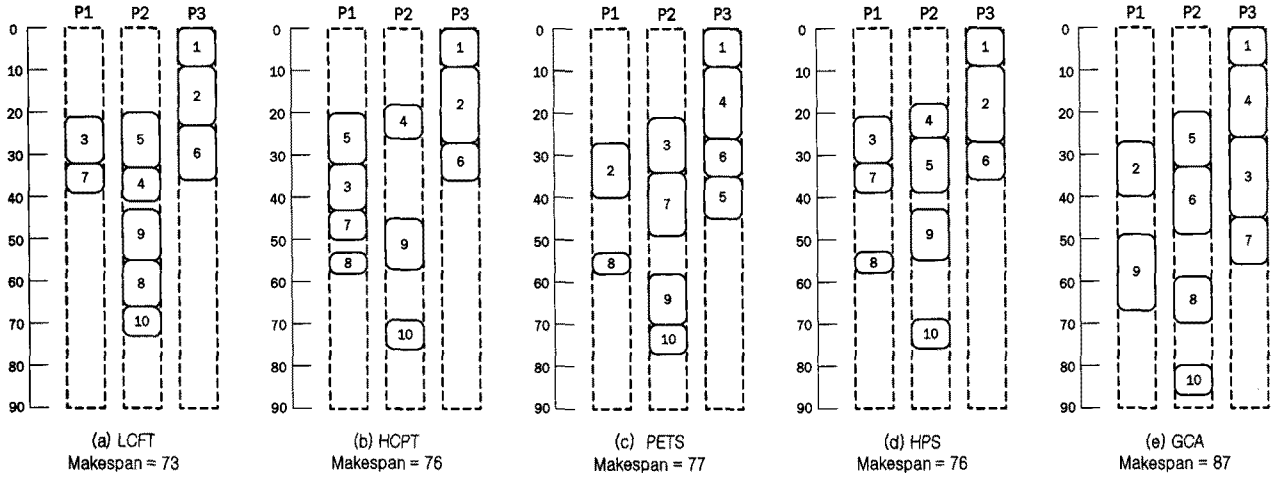

그림 3. LCFT, HCPT, PETS, HPS 그리고 GCA의 전체실행시간
Fig. 3. The makespan generated by LCFT, HCPT, PETS, HPS and GCA.

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

기존의 연구에서 프로세서를 선택하여 태스크를 할당하기 위한 방법은 HEFT, CPOP, PETS, HPS, GCA와 같은 알고리즘에서 사용한 삽입 정책과 HCPT 알고리즘에서 사용한 비 삽입정책으로 나눌 수 있다. 삽입정책은 프로세서에 할당되어진 태스크들 사이에 태스크를 실행할 수 있는 여유공간이 존재하고 선행제약을 만족한다면 태스크를 삽입하는 정책이며 삽입정책이 비 삽입정책에 비해 효율적인 결과를 보인다. 본 논문에서는 기존에 연구되어진 삽입정책을 기반으로 태스크의 실행 완료 시간인 ect가 최소인 프로세서에 태스크를 할당한다. 우선순위 결정 단계에서 정해진 태스크 순서대로 표 1에 표현된 3개의 프로세서에 할당 할 때 시작 태스크 t_s는 $ect(t_s, p_1) = 14$, $ect(t_s, p_2) = 16$, $ect(t_s, p_3) = 9$의 값을 갖게 되므로 최소의 ect을 만족하는 p_3프로세서에 할당된다.

그림 2는 LCFT 알고리즘의 의사코드이다. LCFT의 복잡도는 PETS, HPS 알고리즘과는 같은 $O(v+e)(p+\log v)$의 복잡도를 가진다. 여기서 v는 태스크의 개수, e는 에지의 개수 그리고 p는 프로세서의 개수이다. 레벨 정렬 단계에서 사용된 BFS의 복잡도는 $O(v+e)$의 값을 가지고 우선순위 큐는 이진 힙(Binary Heap)을 사용하였기 때문에 $O(\log v)$의 복잡도를 갖는다. 또한 모든 프로세서 p에 대하여 최소의 실행 완료 시간을 갖는 프로세서를 선택하는데 걸리는 복잡도를 포함하여 전체 복잡도는 $O(v+e)(p+\log v)$을 갖는다.

그림 3은 그림 1의 DAG 그래프와 표 1의 계산비용을 이용하여 본 논문에서 제안한 LCFT 알고리즘과 PETS, HCPT, HPS, GCA 알고리즘의 전체 실행 시간(makespan)를 보여준다. 그림에서 볼 수 있듯이 LCFT는 73의 결과를 얻어 77의 PETS, 76의 HPS, 76의 HCPT 그리고 87의 GCA 보다 성능 향상이 있는 것을 확인 할 수 있다.

V. 성능 분석 및 평가

이 장에서는 본 논문에서 제안한 알고리즘과 기존의 분산 이기종 컴퓨팅 시스템을 위한 스케줄링 알고리즘인 PETS, HPS, HCPT, GCA와의 성능 비교 결과를 기술한다. 알고리즘들의 성능 비교를 위한 다양한 종류의 척도들이 기존 논문에서 제시된 바 있다[4~18]. 공정하고 정확한 비교를 위하여 기존 논문에서 사용한 비교 기준들 중에서 다음의 기준들에 의해 각 알고리즘의 성능을 비교한다.

· Makespan. Makespan은 주어진 입력 그래프 DAG의 수행이 완료된 시간을 의미하며 식 (8)과 같이 마지막 태스크 t_e가 프로세서 p_k에서 실행 완료된 시간으로 표현 될 수 있다.

$$makespan = ect(t_e, p_k) \quad (8)$$

· Normalized Schedule Length(NSL). 알고리즘의 성능 측정은 makespan을 주된 방법으로 이용한다. 하지만 각각 다른 속성을 가진 많은 양의 DAG 그래프들이 이용되기 때문에, Normalized Schedule Length(NSL)라 불리는 하한(lower bound) 값으로

makespan을 표준화 시킨다. 알고리즘의 NSL값은 다음 식으로 정의된다.

$$NSL = \frac{makespan}{\sum_{t_i \in CP} \min_{p_k \in P}\{w_{i,k}\}} \quad (9)$$

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

· SpeedUp. SpeedUp은 순차 실행 시간(sequential execution time)을 병렬 실행 시간(parallel execution time)으로 나눈 값으로 정의 된다. 순차 실행 시간은 모든 태스크들을 계산비용의 합이 최소가 되는 단일 프로세서에 할당하는 방법으로 계산하고 병렬 실행 시간은 makespan을 의미한다. 따라서 SpeedUp이 클수록 성능이 더 좋은 알고리즘으로 볼 수 있다.

$$Speedup = \frac{\min_{p_m \in P}\left\{\sum_{t_i \in T} w_{i,m}\right\}}{makespan} \quad (10)$$

· 알고리즘간의 우열비교. 어떤 알고리즘이 다른 알고리즘에 비해 makespan이 더 좋은 경우의 개수(better), 동일한 경우(equal)의 개수, 나쁜 경우(worse)의 개수를 산출하여 비교하여 직관적인 성능을 비교한다.

1. 입력 그래프 생성

알고리즘의 공정한 성능 비교를 위해 기존의 연구에서 사용된 DAG 그래프를 인용하였으며 표준 태스크 그래프(Standard Task Graph, STDGP)[18][19] 프로젝트에서 제공하는 효율적인 표준 DAG 그래프를 사용하였다. STDGP는 크게 두 가지 집합으로 나눌 수 있다. 첫 번째 집합은 실제 응용 프로그램으로부터 모델링한 robot-control, sparse-matrix, fpppp 3종의 DAG로 이루어진 집합이고 두 번째 집합은 임의로 만든 DAG 집합이다. STDGP에서 STG 포맷으로 제공하는 DAG 집합은 통신비용과 계산비용에 대한 정보는 없고 단지 그래프내의 태스크의 개수와 태스크들의 관계만을 제공한다. 따라서 통신비용, 계산비용이 적용된 DAG를 생성하기 위해 STG 포맷을 자체 변환기를 사용하여 변환하고 다음의 매개변수들을 입력으로 받아 가중치가 적용된 DAG 그래프를 생성하였다.

· 통신비용 대 계산비용의 비율 (CCR). 평균 통신비용에 대한 평균 계산비용의 비율이다. 만일 $CCR \ll 1$이면 이는 연산 집약적인 응용 프로그램이라 할 수 있으며 $CCR \gg 1$이면 통신 시간이 차지하는 비율이 계산 시간에 비해 상대적으로 높다는 것을 의미한다. 본 논문에서는 CCR값을 0.1부터 5까지 변화시키면서 실험을 하였다.

· 프로세서에 대한 이질성 매개변수 (η). 프로세서에서 태스크를 실행하는데 걸리는 실행 시간의 이질성 척도이다. 큰 값을 갖는다면 어떤 태스크를 실행 할 때 프로세서들 간의 계산비용이 많은 차이가 있는 것이고 작은 값을 갖는다면 시스템상의 어떤 프로세서에 할당되어도 태스크의 계산비용이 거의 같다는 것을 의미하는 매개변수이다. 그래프 상에서 각 태스크 t_i의 계산비용의 평균, 즉 $\overline{t_i^w}$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 시스템내의 각 프로세서 p_k에서 각 태스크 t_i의 계산비용은 다음 범위 내에 속하게 된다.

$$\overline{t_i^w} \times (1-\frac{\eta}{2}) \le w_{i,k} \le \overline{t_i^w} \times (1+\frac{\eta}{2}) \quad (11)$$

본 논문에서는 $\eta = \{0.1, 0.5, 1.0, 1.5, 2.0\}$으로 설정하여 실험 하였다.

· DAG 그래프의 태스크들의 수 (t). 기존 논문에서 실험에 사용된 태스크의 개수는 최대 140개 까지만 비교하였으나 본 논문에서는 50, 100, 300, 500, 750개의 태스크가 포함된 DAG 그래프를 이용하여 실험 하였다.

· 프로세서의 개수 (m). 알고리즘의 병렬성을 실험하기 위해 프로세서의 개수를 2, 4, 8, 16, 32개로 변화시키면서 실험을 하였다.

2. 실험 결과

여러 가지 매개변수를 이용하여 생성된 DAG 그래프를 본 논문에서 제안한 LCFT 알고리즘과 HCPT, PETS, HPS, GCA 알고리즘에 적용하여 실험 하였다. 그림 4와 5는 프로세서의 개수(2, 4, 8, 16, 32)마다 임의의 그래프를 각 태스크의 개수(50, 100, 300, 500, 750)당 100개씩 η의 값을 변화 시키면서 생성하여 CCR의 변화에 따른 NSL과 SpeedUp의 평균을 구한 그래프이다.

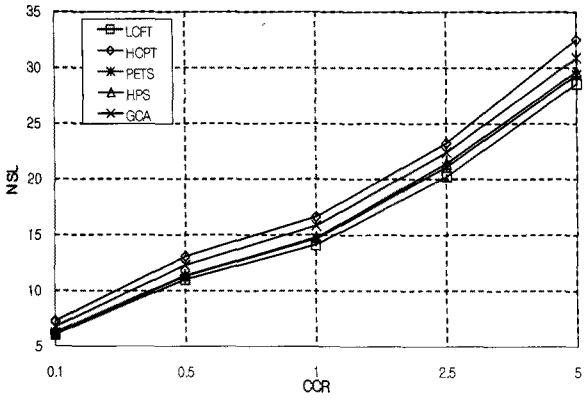

그림 4. *CCR*의 변화에 따른 평균 NSL
Fig. 4. Average NSL for varying *CCR*

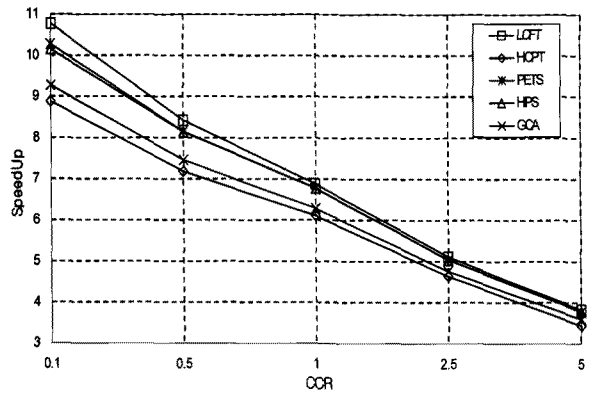

그림 5. *CCR*의 변화에 따른 평균 SpeedUp
Fig. 5. Average SpeedUp for varying *CCR*.

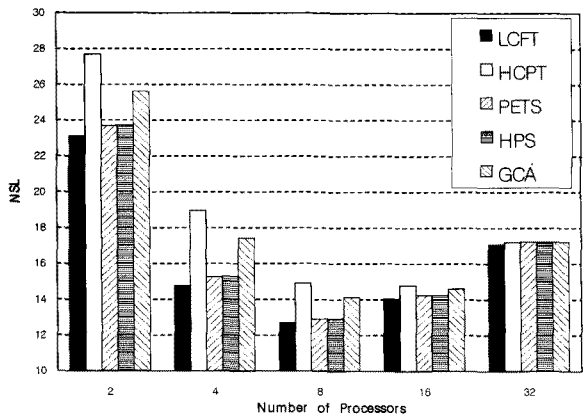

그림 6. 프로세서의 변화에 따른 평균 NSL
Fig. 6. Average NSL for varying Processors.

NSL과 SpeedUp에서 LCFT 알고리즘이 HCPT, PETS, HPS, GCA보다 평균 12.87%, 1.2%, 1.6%, 8%의 성능향상이 있는 것을 확인 할 수 있었다.

그림 6과 7은 프로세서의 변화에 따른 NSL과 SpeedUp의 평균값을 보여주고 있다. 프로세서의 개수가 적을수록 LCFT가 다른 알고리즘에 비해 성능 향상

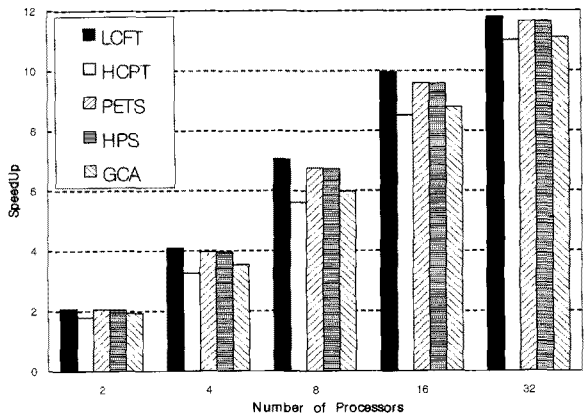

그림 7. 프로세서의 변화에 따른 평균 SpeedUp
Fig. 7. Average SpeedUp for varying Processors

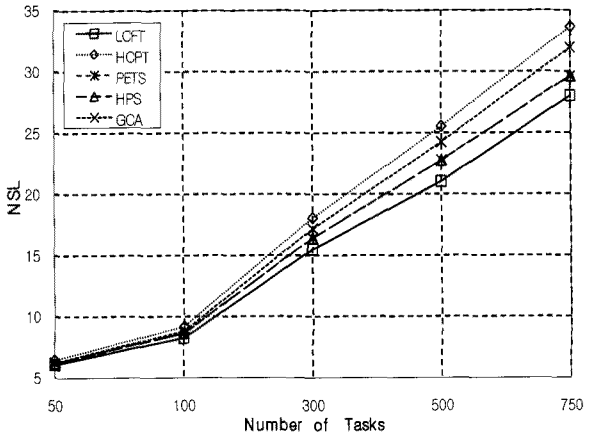

그림 8. 태스크의 변화에 따른 평균 NSL
Fig. 8. Average NSL for varying Tasks.

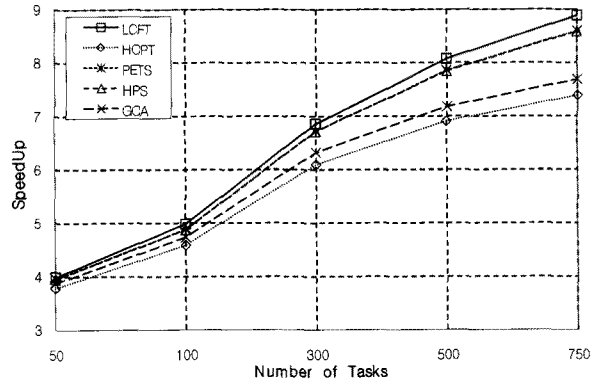

그림 9. 태스크의 변화에 따른 평균 SpeedUp
Fig. 9. Average SpeedUp for varying Tasks.

이 많은 것을 확인 할 수 있다. 프로세서의 개수가 32일 때 다른 알고리즘들은 거의 비슷한 성능을 보여 주고 있지만 LCFT 알고리즘은 NSL과 SpeedUp 모두에서 성능향상이 있는 것을 볼 수 있다.

그림 8과 9는 태스크의 변화에 따른 그래프이다. 태

표 3. 다른 알고리즘과 LCFT의 비교
Table 3. Comparison between LCFT and other algorithms.

		HCPT	PETS	HPS	GCA	Combined
LCFT	better	44782	42797	43667	45603	70.74%
	equal	3308	6525	5540	3346	7.49%
	worse	14410	13178	13293	13551	21.77%

스크의 수가 100개 보다 적을 때보다 태스크의 개수가 많을 때 즉 병렬 프로그램의 크기가 클수록 LCFT 알고리즘의 성능이 다른 알고리즘에 비해 높다는 것을 알 수 있다.

표 3은 프로세서의 개수, 태스크의 개수, η의 값 그리고 *CCR* 값을 변화시켜 각 알고리즘 당 62500개 그래프를 실험하여 LCFT 알고리즘의 makespan이 더 좋은 경우(better), 같은 경우(equal), 나쁜 경우(worse)의 개수를 표현한 것이다. LCFT 알고리즘이 4개의 다른 알고리즘보다 70.74% 더 좋은 결과를 보였으며 나쁜 경우는 21.77%가 나오는 것을 확인 할 수 있다.

VI. 결 론

본 논문에서는 주어진 DAG 그래프의 태스크들을 레벨화 하고 태스크의 실행시간, 부모태스크로부터 데이터를 전달받는데 걸리는 통신비용, 가장 비용이 높은 자식태스크를 가지는 태스크에 우선순위를 높게 주면서 기존의 알고리즘보다 같거나 작은 복잡도를 가지는 새로운 레벨화된 리스트 스케줄링 알고리즘인 LCFT를 제안하였다. 또한 제안한 알고리즘의 공정한 성능을 확인하기 위해 기존의 논문에서 연구한 DAG 그래프를 인용하고 다양한 매개변수를 이용하여 실험 하였다. 기존의 실험에서 사용된 태스크의 개수보다 많은 750개의 태스크와 프로세서의 종류를 다양화 하여 실험함으로서 큰 크기의 병렬 프로그램과 분산 이기종 컴퓨팅 환경을 고려하였다. 실험 결과 기존에 연구된 리스트 스케줄링 알고리즘 보다 월등히 좋은 성능을 보였으며 특히 태스크의 개수가 많고 *CCR*값이 클 수록 다른 알고리즘보다 더 좋은 성능이 나오는 것을 확인 할 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] 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.

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

[6] 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.

[7] Kafil, M. and I. Ahmed, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, 6: 42-51, 1998.

[8] Ranaweera, A. and D.P. Agrawal, "A task duplication based algorithm for heterogeneous systems," *Proc. IPDPS*, pp: 445-450, 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] 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.

[11] 윤완오, 윤정희, 이창호, 김효기, 최상방, "분산 이기종 컴퓨팅 시스템에서 효율적인 리스트 스케줄링 알고리즘," *전자공학회 논문지*, 제46권 CI편, 제3호, 86-95쪽, 2009년 5월.

[12] 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.
[13] H. Togcuglou, S, Hariri and M.Y. Wu, "Task Scheduling Algorithms for Heterogeneous Processor," *Proceeding of the HCW*, pp.3-14, 1999.
[14] 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.
[15] 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.
[16] 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.
[17] E. Ilavarasan, P. Thambidurai and R. Mahilmannan, "High Performance Task Scheduling Algorithm for Heterogeneous Computing System," *LNCS 3718*, pp. 193-203, 2005.
[18] Takao Tobita and Hironory kasahara, "A Standard Task Graph Set for Fair Evaluation of Multiprocessor Scheduling Algorithms," *Journal of Scheduling*, 5, pp. 379-394, 2002.
[19] 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, 컴퓨터 구조, 임베디드 시스템, 컴퓨터 네트워크>

윤 준 철(학생회원)
2006년 2월 한국산업기술대 전자공학과 졸업.
2008년 2월 인하대학교 전자공학과 석사 졸업.
2009년~현재 인하대학교 전자공학과 박사 과정.
<주관심분야 : 병렬 및 분산 처리 시스템, 컴퓨터 구조, 컴퓨터 네트워크>

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

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