이질 시스템에서 통신 시간을 고려한 효율적인 복제 기반 태스크 스케줄링

준회원 윤 완 오*, 백 정 규*, 신 광 식*, 정 진 하*, 종신회원 최 상 방*

Efficient Duplication Based Task Scheduling with Communication Cost in Heterogeneous Systems

Wan-Oh Yoon*, Jueng-Kuy Baek*, Kwang-Sik Shin*, Jin-Ha Cheong* ***Associate Members***,
Sang-Bang Choi* ***Lifelong Member***

요 약

스케줄링의 목적은 입력 작업(DAG)에 대한 스케줄 결과 길이를 최소화 하는 것이다. 이런 스케줄링 문제는 잘 알려진 '정해진 시간 내에 해결하기 어려운 문제(NP-complete)'이며 최적의 스케줄링 결과 값을 얻기 위해서는 휴리스틱으로 해결해야 한다. 선후 관계의 제약을 갖는 노드들의 스케줄링을 효율적으로 수행하기 위해 부모 노드와 이질 프로세서에 대한 정보를 고려하는 TANH(the Task duplication based scheduling Algorithm for Network of Heterogeneous systems), GDL, BIL, TDS과 같은 많은 알고리즘이 제안되었다. 본 논문은 기존의 TANH 스케줄링에서 나타나는 여러 개의 부모 노드와 이질 프로세서에 대한 다양한 경우를 충분히 고려하지 못한 점을 보안하여 향상된 스케줄링을 수행할 수 있는 DTSC (Duplication based Task Scheduling with Communication Cost in Heterogeneous Systems)알고리즘을 제안하였다. 제안된 알고리즘의 성능은 기존 TANH, GDL 알고리즘과 비교하였으며, 스케줄링의 성능 향상을 보여 주었다.

Key Words : Heterogeneous System, Task Scheduling, Duplication, Heuristic, DAG

ABSTRACT

Optimal scheduling of parallel tasks with some precedence relationship, onto a parallel machine is known to be NP-complete. The complexity of the problem increases when task scheduling is to be done in a heterogeneous environment, where the processors in the network may not be identical and take different amounts of time to execute the same task. This paper introduces a Duplication based Task Scheduling with Communication Cost in Heterogeneous Systems (DTSC), which provides optimal results for applications represented by Directed Acyclic Graphs (DAGs), provided a simple set of conditions on task computation and network communication time could be satisfied. Results from an extensive simulation show significant performance improvement from the proposed techniques over the Task duplication-based scheduling Algorithm for Network of Heterogeneous systems(TANH) and General Dynamic Level(GDL) schedluling algorithm.

Ⅰ. 서 론

최근 디지털 영상처리, 컴퓨터 비전, 일기 예보를 위한 모델링, 유체의 흐름, 이미지 처리, 리얼 타임 시스템, 멀티미디어 정보처리 및 통신 분야 등에서는 대규모의 데이터베이스와 그 처리 알고리즘의

※ 본 논문은 건설교통부 항공선진화사업의 연구비지원(과제번호#07항공-항행-03)에 의해 수행 되었습니다.
* 인하대학교 전자공학과 컴퓨터 구조 및 네트워크 연구실(wanoh38@inha.ac.kr)
논문번호 : KICS2007-08-386, 접수일자 : 2007년 8월 29일, 최종논문접수일자 : 2008년 2월 11일

복잡성 때문에 병렬 고속연산이 절실히 요구되고 있으며, 이를 위해 병렬 처리 및 분산 시스템이 주목 받고 있다. 그림 1은 서버 역할을 하는 머신이 각각의 머신에 태스크를 할당하고 정보를 교환할 수 있도록 네트워크로 연결된 병렬 및 분산 시스템의 한 예이다. 이러한 시스템은 시스템간의 처리된 데이터 및 정보 교환을 위해서 네트워크 의존성이 강하다. 또한 병렬 및 분산 시스템은 시스템과 프로세서의 특성이 다른 통신 시간 및 처리 환경을 갖는 이질 시스템이다. 따라서 이러한 이질 시스템을 효율적으로 다루기 위해서는 통신 시간과 작업 수행시간이라는 요소가 필요하다. 스케줄링에서 노드의 선후 관계 정보를 교환하기 위해서 사용되는 시스템간의 통신 시간과 태스크의 수행 시간은 이질 시스템 내에서의 태스크 스케줄링의 중요한 요소이다. 마이크로프로세서 및 다양한 처리 기술의 발전에 힘입어 시스템 및 프로세서의 성능 향상으로 태스크 처리 시간을 단축시킬 수 있다. 하지만 이에 비해 통신 시간의 감소는 이루어지지 못했다. 또한 이질 시스템 환경으로 인한 작업 수행 시간의 차이 역시 병렬 처리 및 분산 시스템의 중요한 요소이다. 수행 시간의 차이는 프로세서 및 시스템간의 성능 차이에서 발생하는 문제점이다. 이런 문제점을 해결하기 위한 방법으로 통신 시간의 감소를 위해 선후 관계를 갖는 노드를 동일 프로세서에 할당 하는 방법과 부모 노드의 복제를 이용해 노드간의 통신 시간을 감소시키는 많은 알고리즘이 제안되었고, 이질 시스템 환경에서 발생하는 수행 시간의 차이는 태스크를 처리하기 위해 최소의 시간이 소요되는 프로세서 및 시스템에 할당하는 알고리즘이 제안되었다. 스케줄링의 목적은 입력 그래프에 대한 선후 관계를 고려하여 스케줄링 결과 길이를 최소화하는 것이다.

최적의 스케줄링은 병렬 처리 및 분산 시스템의 성능 향상을 가져다주지만, 이런 스케줄링 문제는 다항 시간(polynomial time)내에 결과를 얻을 수 없는 NP-complete[1-3]로서 최적의 스케줄링 결과를 얻기 위해서는 모든 경우를 고려해야 한다. 최근 최적의 스케줄링을 위해 활발히 연구되고 있는 분야는 크게 리스트 스케줄링, 클러스터 스케줄링, 태스크 복제 스케줄링을 들 수 있다. 태스크 복제 스케줄링은 노드들 간의 상호 의존성에 의한 통신 시간을 감소시키기 위해 선행 노드의 복제를 통해 스케줄링을 수행한다. 태스크 복제 스케줄링은 어떤 노드를 복제하느냐에 따라 다양한 알고리즘이 존재 한다. 이런 스케줄링은 다른 스케줄링에 비해 복잡도가 크다[3].

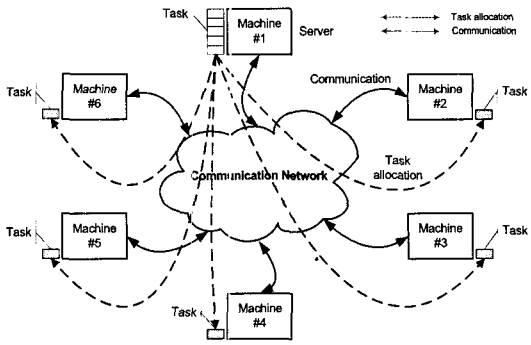


그림 1. 병렬 및 분산 시스템 예.

본 논문에서 제안한 알고리즘은 태스크 복제를 기반으로 한 클러스터 스케줄링이다. 스케줄링의 최대 관점인 노드의 선후 관계를 충분히 고려할 수 있는 알고리즘을 제안하고, 이를 이용해 1개의 부모 노드가 있는 경우, 2개 이상의 부모 노드가 있는 경우로 나누어 해당 노드가 최적의 프로세서에서 최소의 수행 시간을 갖는 것을 수식적으로 나타내었다. 이 계산 값을 바탕으로 최적 부모 노드를 기준으로 클러스터를 구성한다. 구성된 클러스터의 최적 프로세서는 클러스터를 구성하는 노드의 수행 시간 합이 최소가 되는 프로세서를 선택한다. 클러스터의 최적 프로세서로 선택된 프로세서를 기준으로 노드간의 선후 관계를 해결하기 위해 메시지 스케줄링을 수행한다.

제안된 스케줄링의 성능 분석은 통신 시간과 수행 시간의 비인 *CCR*(communication cost computation cost ratio)과 노드의 수의 변화를 바탕으로 스케줄링 간의 성능 향상을 비교했다. 또한 프로세서가 사용되는 시간인 프로세서의 작업 시간(busy time)을 바탕으로 프로세서의 효율을 비교하였다. 프로세서의 작업 시간이 적을수록 다른 태스크가 프로세서를 사용할 수 있는 시간이 많아지므로 프로세서의 효율을 알 수 있다. 다음으로 입력 그래프(Directed Acyclic Graph)에 대한 작업이 빨리 끝난 것을 나타내는 클러스터가 할당된 프로세서의 완료 시간(finish time)을 비교하였다. 마지막으로 시간에 따른 프로세서의 개수를 통해 전체적으로 적은 프로세서를 사용해 스케줄링을 수행 하는지를 비교하였다.

본 논문은 다음과 같이 구성되어 있다. Ⅱ장에서는 이질 시스템 환경에서 기존의 태스크 스케줄링에 대해 알아보고, 기존 스케줄링의 문제점에 대해

서 설명한다. Ⅲ장에서는 제안된 스케줄링의 전체적인 수행 과정과 알고리즘에 대해 설명하고, 각 과정별로 수식적으로 정리한다. Ⅳ장에서는 제안된 알고리즘과 기존 알고리즘간의 결과를 다양한 측면에서 비교 한다. 마지막으로 Ⅴ장에서는 본 논문의 연구 성과와 앞으로의 연구 과제를 제시한다.

Ⅱ. 관련 연구

2.1 입력 그래프의 정의

분할된 태스크의 선후 관계는 입력 그래프인 DAG로 표현 할 수 있다. 그림 2는 입력 그래프의 한 예로서, 그래프상의 원은 노드를 의미하고 화살표는 부모 노드와 자식 노드간의 선후 관계를 나타내고 있다. 마지막으로 화살표 상의 숫자는 두 노드간의 통신 시간을 의미 한다. 입력 그래프는 (V, E, W, EXC, CMC)로 표현되며, V는 스케줄링을 하기 위한 모든 노드 혹은 태스크를(본 논문에서 노드(*node*)와 태스크(*task*)는 동일한 뜻으로 간주한다.) 나타내고, E는 프로세서간의 통신 시간을 나타낸다. W는 노드를 할당 받을 수 있는 프로세서를 나타내고, EXC는 수행 시간(execution cost)으로 프로세서에 따른 노드의 수행 시간을 의미한다. 즉 $EXC(j,x)$는 $x(x\in W)$에서 $j(j\in V)$ 노드의 수행 시간을 나타낸다. 마지막으로 CMC는 통신 시간(communication cost)으로 두 노드간의 통신 시간을 의미한다. 따라서 $CMC(j,k)$는 노드 j와 k가 $j,k\in V$을 만족 할 때 두 노드 사이의 통신 시간을 의미한다. 만약 노드 j와 k가 동일한 프로세서 $x(x\in W)$에 할당 되어 있다면 두 노드간의 통신 시간 $CMC(j,k)$ = 0이 된다. 또한 각 노드의 실행은 비선점(non-preemptive) 특성을 갖는다. 이질 시스템 환경에서 프로세서 혹은 머신간의 통신 채널 및 대역은 충분하다고 가정하고, 통신 구성(communication topology)은 완전 연결(fully connected)로 가정한다. 그림 2의 노드 1처럼 부모 노드가 존재하지 않는 노드를 시작 노드(entry node)라 하고, 노드 13처럼 자식 노드가 존재하지 않은 노드를 마지막 노드(exit node)라 정의 한다. 스케줄링을 위해서는 오직 한 개의 입력 노드와 출력 노드가 필요하다. 만약 2개 이상의 입력 노드와 출력 노드가 있다면 이 노드들을 묶을 수 있는 의사 노드(pseudo node)를 추가할 수 있으며 의사 노드의 수행 시간과 통신 시간은 모두 0 값을 가지고 이들 노드는 스케줄링 상에 아무런 영향을 주지 않는다고 가정한다[3]-[6],[9-10].

2.2 스케줄링에 필요한 파라메타의 정의

본 논문에서 태스크 스케줄링을 수행하기 위해서 필요한 각 노드의 est(earliest start time), ect (earliest completion time), $fpred$(favorite predecessor), sep(shortest execution processor)를 다음과 같이 정의하며 여기에서 노드 j,k,l은 $\{j,k,l\}\subset V$이고, $\{k,l\}\subset PRED(j)$을 만족한다고 가정한다. 즉 $PRED(j)$는 노드 j의 부모 노드의 집합이다.

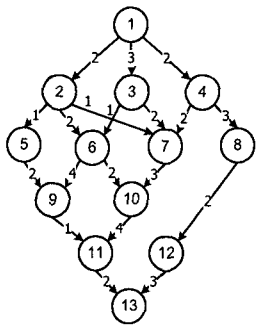


그림 2. 노드의 선후 관계를 나타내는 입력 그래프

- $est(j)$: 노드 j가 해당 프로세서에서 가장 빨리 시작할 수 있는 시간이며 $est(j/x)$는 프로세서 $x(x\in W)$에서 노드 $j(j\in V)$가 가장 빨리 시작할 수 있는 시간을 의미한다.
- $ect(j)$: 노드 j가 해당 프로세서에서 가장 빨리 완료할 수 있는 시간이며 $ect(j/x)$는 프로세서 $x(x\in W)$에서 노드 $j(j\in V)$가 가장 빨리 완료할 수 있는 시간을 의미한다. 이질 시스템이므로 노드의 수행 시간은 프로세서 마다 다르다. 따라서 노드 j가 부모 노드 k와 동일 프로세서에 있는 경우와 노드 j의 수행 시간이 가장 짧은 프로세서에 있는 경우를 비교해서 최소가 되는 것이 $ect(j)$가 된다.
- $fpred(j)$: 노드 j의 부모 노드(predecessor node) 집합을 $PRED(j)$로 정의하고, $fpred(j)$는 부모 노드 중에서 노드 j에 가장 늦게 데이터를 보내주는 노드를 의미하며 최적 부모 노드라 한다. 노드 j에 정보를 늦게 보내주는 순서에 따라 $fpred_1(j), fpred_2(j), fpred_3(j), \ldots$ 로 표현한다.
- $fproc(j)$: 노드 j의 $fproc(j)$는 $k\in PRED(j)$일 때 부모 노드 k을 고려하여 자식 노드 j가 할당 될 때 최소의 완료 시간(completion time)을 갖는 프로세서를 의미한다.
- $sep(j)$: $sep(j)$는 노드 j의 수행 시간이 가장 짧은 프로세서를 의미하며 최적 프로세서라 한다.

$sep(j/p)$는 프로세서 p에서 노드 j의 수행 시간을 의미한다. 노드 j의 수행 시간이 짧은 순으로 $sep_1(j), sep_2(j), sep_3(j), \ldots$로 표현한다.

2.3 기존 스케줄링의 문제점

클러스터 스케줄링의 대표적인 TANH 알고리즘은 4 단계로 이루어져 있다. 첫 번째 단계는 입력 그래프에 대해 $est, ect, fpred, fproc, level, queue$을 하향식(top-down)으로 계산한다. 두 번째 단계는 $last$ (latest start time), $lact$(latest completion time)를 상향식(bottom-up)으로 계산한다. 첫 번째, 두 번째 단계에서 계산된 값은 다음 단계의 클러스터 구성을 위한 정보가 된다. 또한 이 두 단계는 스케줄링에서 가장 중요한 크리티컬 패스(critical path)를 찾아 최적화 하는 과정과 유사하다. 다음으로 노드의 레벨은 마지막 노드(exit node)로부터 수행 시간의 합이 가장 큰 값을 의미한다. 클러스터를 구성하기 위해 사용되는 큐(queue)에서 노드들의 순서는 레벨 값의 오름차순에 의해 정해진다. 세 번째 단계는 앞에서 구한 결과 값을 이용해 노드 간의 선후 관계를 바탕으로 클러스터를 구성한다. 계산된 각 노드의 $fpred$에 의해 클러스터를 구성하게 된다. 만약 이전 클러스터 구성 시 노드 j의 $fpred_1(j)$가 할당 되었다면 노드 j가 할당된 프로세서에서 가장 적은 수행 시간을 갖고, 아직 할당되지 않은 부모 노드를 선택함으로서 클러스터를 구성한다. 구성된 클러스터는 클러스터의 첫 번째 노드의 최적 프로세서에 할당한다. 마지막 단계에서 노드 간의 메시지 스케줄링을 수행하면 스케줄링이 완성 된다[10]. 프로세서 $\{x, y, z\} \subset W$, 노드 $\{k, l, m\} \subset PRED(j)$, $fproc(j) = p$인 경우 TANH 알고리즘의 $est(j/p)$계산식은 아래와 같다.

$$est(j/p) = \underset{l,k \in PRED(j)}{Min}[\underset{k \neq l, fp(k) = w}{Max}((est(k) + \tau(k,w)), \quad (1)$$
$$(ect(l) + CMC(l,j))]$$
$$k \neq l, fp(k) \neq w \text{ where } w = fp(j)$$

노드 j의 $est(j)$는 부모 노드인 k에 의해 좌우된다. 또한 부모 노드 k의 $est(k)$ 역시 노드 k의 부모 노드에 의해 결정된다. 그러나 이미 노드 k에 대한 $est(k)$, $ect(k)$가 결정되고 $fproc(k)$가 결정된 상태인데도 식 (1)은 자식 노드 j가 $fproc(j)$에 할당되는 상황에서 부모 노드 k가 $fproc(j)$에 할당되는 경우와 다른 프로세서에 할당되는 경우를 비교해 $est(j)$을 결정한다. 이것은 앞서 부모 노드 k에 대해 계산된 값을 의미 없게 만드는 것이다. 따라서 각 노드의 $est, ect, fproc$을 결정짓기 위해서는 하향식(top-down)으로 계산된 각 노드의 부모 노드에 대한 $est, ect, fproc$ 값을 적용해야 한다.

$fpred(j) = k$이면 통신 시간을 줄이기 위해 노드 k와 노드 j을 동일 프로세서에 할당해야 하며 이 경우 $est(j) = ect(l) + CMC(l,j)$가 된다. 그러나 이질 시스템의 특성에 의해 $ect(l) + CMC(l,j)$보다 프로세서 y에서 실행되는 노드 k의 실행시간이 더 큰 경우가 발생될 수 있다. 따라서 모든 조건을 만족하는 $est(j)$ 계산식은 아래와 같이 재정의 되어야 한다. 여기서 $2^{nd}Max$란 두 번째 큰 값을 의미한다.

$$est(j) = Max[2^{nd}Max[(ect(i_1) + CMC(i_1,j)), \quad (2)$$
$$(ect(i_2) + CMC(i_2,j)),$$
$$(ect(i_3) + CMC(i_3,j)), \cdots], ect(fpred_1(j))]$$
$$\{i_1, i_2, i_3, \ldots\} \subset PRED(j)$$

즉, $est(j)$는 각 부모 노드의 $ect(i) + CMC(i,j)$ 값 중에서 두 번째로 큰 값과 $fpred(j)$의 $ect(fpred(j))$ 값 중에서 큰 값을 선택하면 된다.

TANH에서 $ect(j)$는 다음과 같이 정의한다. 여기서 $EXC(j, fproc(j))$는 j의 $fproc$에서 노드 j의 수행 시간을 의미한다.

$$ect(j) = est(j) + EXC(j, fproc(j)) \quad (3)$$

이질 시스템에서 노드의 수행 시간은 프로세서마다 다르기 때문에 단순히 $est(j)$에 노드 j의 $fproc(j)$에서의 수행 시간을 더해 $ect(j)$을 구하는 것은 적절한 계산이 아니다. $est(j)$는 부모 노드에 의해 결정되므로 노드 j가 자신의 $fproc(j)$에 할당되는지 보장할 수 없기 때문이다. $est(j)$을 제공하는 k가 할당된 프로세서 p에 노드 j을 할당하는 경우뿐만 아니라 노드 j 자신의 $fproc(j)$에서 통신 시간을 고려해도 더 빨리 완료할 수 있는 경우도 발생할 수 있기 때문이다. 따라서 이 두 값을 비교해 최소값을 $ect(j)$로 선택해야 한다. 위의 두 경우를 만족하는 $ect(j)$의 식은 다음과 같이 정의 할 수 있다.

$$ect(j) = Min[(est(j) \quad (4)$$
$$+ EXC(j, k\text{가할당된 프로세서}p)),$$
$$(est(j)Max + EXC(j, fproc(j)))]$$

마지막으로 구성된 클러스터를 프로세서에 할당하는 기존 방법은 클러스터 시작 노드의 최적 프로세서($fproc$)에 할당하는 것이다. 다음 식은 TANH 알고리즘의 $fproc(j)$을 선택하는 식이다.

$$fproc(j) = p \in W \quad (5)$$
$$est(j/p) + EXC(j,p) < est(j/p) + COMC(k,j) + EXC(j,q),\ q \in W$$

그러나, TANH 알고리즘의 $fproc$ 정의는 부모 노드와 자식 노드간의 관계만을 고려한 것이다. 따라서 시작 노드의 최적 프로세서가 클러스터의 최적 프로세서임을 보장할 수 없다. 본 논문에서는 기존 알고리즘의 문제점을 보안하고 부모 노드와의 선후 관계를 고려하는 새로운 스케줄링 기법을 제안한다.

Ⅲ. 제안된 DTSC 스케줄링 알고리즘

3.1 DTSC 스케줄링

본 장에서는 이질 시스템 환경에서 새롭게 제안된 통신 시간을 고려한 복제 기반 태스크 스케줄링 DTSC에 대해 설명한다. 본 논문에서 제안 하는 DTSC 스케줄링은 4 단계로 구성되어 있다. 첫 번째 단계는 하향식으로 est, ect, $fpred$, sep을 입력 그래프와 각 프로세서에 의한 노드의 수행 시간을 바탕으로 계산한다. 두 번째 단계는 구해진 $fpred$ 정보를 고려해 클러스터를 구성하며 클러스터 구성 시 해당 노드의 $fpred$ 우선순위에 따라 구성한다. 세 번째 단계에서는 구성된 클러스터의 최적 프로세서를 찾고 그 프로세서에 노드를 할당한다. 마지막으로 노드간의 메시지 스케줄링을 수행한 후 빈 슬롯을 이용해 태스크 복제를 수행하면 최종 스케줄링이 완성된다. DTSC 스케줄링 알고리즘의 각 스텝 수행 과정의 세부 내용은 다음과 같다.

3.1.1 단계 1 - est, ect, $fpred$, sep 계산 방법

est, ect, $fpred$, sep의 계산은 현재 노드의 부모 노드가 1개 있는 경우와 2개 이상 있는 경우로 나누어서 생각한다. 부모 노드의 개수에 관계없이 sep의 우선순위는 수행 시간이 가장 짧은 프로세서의 순서이다. 노드 j을 할당할 수 있는 프로세서 p_1, p_2가 있는 경우 $sep(j/p_1) = 2$, $sep(j/p_2) = 5$이면 $sep_1(j) = p_1$이 된다. 만약 $sep(j/p_1) = 2$, $sep(j/p_2) = 2$이면 프로세서의 순서에 따라 $sep_1(j) = p_1$ 되고 $sep_2(j) = p_2$가 된다. 즉 수행 시간이 동일할 경우는 주어진 프로세서의 순서를 따른다.

(1) 1 개의 부모 노드만 있는 경우

한 개의 부모 노드만 있는 경우 부모 노드와 자식 노드를 다른 프로세서에 할당하면 통신 시간이 추가 된다. 두 노드가 각각 다른 프로세서에 할당되었을 경우의 전체 완료시간이 동일 프로세서에 할당되었을 경우의 전체 완료시간보다 크다면 통신 시간을 줄이기 위해 한 개의 부모 노드가 있는 경우에는 동일 프로세서에 할당해야 한다. 다음 식은 노드 j의 $fpred(j)$인 노드를 선택하는 식이다.

$$fpred(j) = \underset{k \in PRED(j)}{Max}[ect(k)] \quad (6)$$

$est(j)$ 역시 부모 노드에 의해 좌우 된다. 따라서 $est(j)$ 식은 다음과 같이 표현 할 수 있다.

$$est(j) = \underset{k \in PRED(j)}{2^{nd}Max}[(ect(k)] \quad (7)$$

식 (7)의 $2^{nd}Max$는 부모 노드의 ect 값 중에서 두 번째 큰 부모 노드의 ect 값을 의미한다. $2^{nd}Max$는 한 개의 부모 노드가 있는 경우에는 의미가 없다. 다음 절에서 설명할 2개 이상의 부모 노드가 있는 경우에 사용된다. 여러 개의 부모 노드가 있는 경우 부모 노드의 데이터 전송 시간이 가장 큰 값을 $est(j)_{Max}$라 한다. 하지만 한 개의 부모 노드가 있는 경우는 $est(j)$와 동일하며 다음과 같이 표현할 수 있다.

$$est(j)_{Max} = \underset{k \in PRED(j)}{Max}[(ect(k))] = est(j) \quad (8)$$

부모 노드가 한 개 있는 경우는 est을 계산하는 데 많은 경우의 수가 발생하지 않는다. 단순히 한 개의 부모 노드에 대해서만 고려하면 된다. 또한 부모 노드 k와 자식 노드 j가 동일한 프로세서에 할당 되었다면 $fpred_1(j)$, $est(j)$ 계산식에서 부모 노드가 복제되므로 통신 시간이 필요하지 않다.

ect는 부모 노드 k와 자식 노드 j을 동일 프로세서 p에 할당하는 경우와 부모 노드 k는 프로세서 p에 할당되고 자식 노드 j는 자신의 $sep_1(j)$에 할당하는 경우가 있기 때문에 est와 달리 통신 시간을 고려해야 한다. 자식노드와 부모노드가 서로 다른 프로세서에 할당 되는 경우는 두 노드간의 통신 시간이 추가 된다. 프로세서 마다 수행 시간이 다르므로 $ect(j/p)$보다 $ect(j/sep_1(j))$이 작을 경우가 발생할 수 있다. 따라서 두 가지 경우를 비교하여 최소

값을 $ect(j)$로 선택한다. 다음은 두 경우의 최소값을 취하는 $ect(j)$의 계산식이다.

$$ect(j) = Min[(est(j) + EXC(j,k가 할당된 프로세서에)), (est(j)_{Max} + CMC(k,j) + EXC(j,sep_1(j)))] \quad (9)$$
$$k \in PRED(j)$$

한 개의 부모 노드만 있는 경우에는 식 (6)에 의해 $ect(j)$와 $ect(j)_{Max}$는 동일하다. 식 (9)의 두 요소인 $est(j)_{Max} + CMC(k,j) + EXC(j,sep_1(j))$이 $est(j) + EXC(j,p)$보다 클 경우 두 노드는 동일 프로세서에 할당 되고, $est(j) + EXC(j,p)$이 $est(j)_{Max} + CMC(k,j) + EXC(j,sep_1(j))$보다 클 경우에 두 노드는 다른 프로세서에 할당 된다.

(2) 2개 이상의 부모 노드가 있는 경우

$\{p,w,x\} \subset W$, $\{k,l,m\} \subset PRED(j)$일 때 $fpred_1(j) = k$, $fpred_2(j) = m$, $fpred_3(j) = l$이라고 가정한다. 자식 노드 j는 부모 노드 k, l, m이 데이터를 전송해 주지 않으면 수행 할 수 없다. 따라서 자식 노드 j의 est을 결정짓는 노드는 $fpred_1(j)$인 k가 된다. $fpred$의 우선순위는 클러스터를 구성하는데 중요한 역할을 한다. $fpred_1(j) = k$인 부모 노드 k와 자식 노드 j가 다른 클러스터에 구성되면 가장 긴 데이터 전송 시간을 갖게 되므로 통신 시간을 줄이기 위해 $fpred_1(j)$정보를 이용해 클러스터를 구성해야한다. 2개 이상의 부모 노드가 있는 경우 자식 노드의 est, ect는 어떤 부모 노드와 자식 노드가 동일한 프로세서에 할당 되어 있는지와 나머지 다른 부모 노드와의 관계에 따라 결정된다. 부모 노드의 ect와 통신 시간을 더한 값 중 가장 큰 값을 갖는 노드를 $fpred_1(j)$로 선택한다. 다음으로 큰 값을 $fpred_2(j)$로 선택 한다. $fpred$의 선택 방법은 1개의 부모 노드가 있는 경우와 동일하다. 부모 노드들 중에서 데이터 전송 시간이 가장 큰 노드를 선택하면 된다. $fpred$ 계산식은 부모 노드가 한 개 있는 경우의 식 (6)에 부모 노드를 추가하고 모든 부모 노드 중에서 가장 큰 값을 갖는 노드를 선택하면 된다. 다음은 n개의 부모 노드를 갖는 노드 j의 $fpred_1(j)$을 선택하는 식이다.

$$fpred(j) = Max[(ect(i_1) + CMC(i_1,j)), (ect(i_2) + CMC(i_2,j)), \ldots, (ect(i_n) + CMC(i_n,j))] \quad (10)$$
$$\{i_1,i_2,\ldots,i_n\} \subset PRED(j),\ n\ is\ predecessor\#\ of\ node\ j$$

est는 부모 노드가 한 개 있는 경우의 확장 형태와 유사하다. 하지만 2개 이상의 부모 노드가 있는 경우는 각 노드의 ect와 통신 시간의 합을 고려해야 한다. 이중 가장 큰 값을 갖는 노드를 우린 $fpred_1(j)$라 정의 했다. 따라서 최소의 완료 시간을 갖기 위해서는 통신 시간을 줄이기 위해 $fpred_1(j)$인 부모 노드 k와 자식 노드 j을 동일한 프로세서에 할당해야 한다. 만약 $fpred_1(j)$인 부모 노드 k와 자식 노드 j을 각각 다른 프로세서에 할당 한다면 실제 $est(j)$는 노드 j가 자신의 $fpred_1(j)$와 다른 프로세서에 할당 될 경우 노드 j가 가장 늦게 데이터를 전송받는 시간 $est(j)_{Max}$와 같아지게 되므로 $est(j)_{Max} - est(j)$ 만큼의 시간적 지연이 발생한다. 따라서 $fpred_1(j)$인 부모 노드 k와 자식 노드 j는 동일한 프로세서에 할당 되어야 한다. 다음 식은 2개 이상의 부모 노드가 있는 경우 $est(j)$의 계산식이다.

$$est(j) = Max[2^{nd}Max[(ect(i_1) + CMC(i_1,j)), (ect(i_2) + CMC(i_2,j)), (ect(i_n) + CMC(i_n,j))], ect(fpred_1(j))] \quad (11)$$
$$\{i_1,i_2,\ldots,i_n\} \subset PRED(j),\ n\ is\ predecessor\#\ of\ node\ j$$

식 (11)에서 $ect(fpred_1(j))$는 프로세서에 따라 노드의 수행 시간이 다르므로 노드 k의 수행 시간이 노드 l의 $ect(l) + CMC(l,j)$보다 클 경우가 발생 할 수 있기 때문에 추가된 항목이다. $fpred_1(j)$ 노드와 자식 노드 j의 sep_1이 다른 경우 최소의 $ect(j)$을 얻기 위해 노드 j가 $fpred_1(j)$ 노드와 동일 프로세서에 할당되었을 경우와 노드 j가 $sep_1(j)$에 할당되었을 경우를 비교해 작은 값을 $ect(j)$로 선택하게 된다. 노드 j가 $sep_1(j)$에 할당되면 $est(j)_{Max}$ 시점에 노드 j가 수행될 수도 있다. $ect(j)$을 계산하기 위해 필요한 $est(j)_{Max}$는 다음 식과 같으며 여기서 n은 노드 j의 부모 노드의 수다.

$$est(j)_{Max} = Max[(ect(i_1) + CMC(i_1,j)), (ect(i_2) + CMC(i_2,j)), \ldots, (ect(i_n) + CMC(i_n,j))] \quad (12)$$
$$\{i_1,i_2,\ldots,i_n\} \subset PRED(j),\ n\ is\ predecessor\#\ of\ node\ j$$

$ect(j)$는 $fpred_1(j)$와 노드 j가 동일 프로세서에 할당된 경우와 서로 다른 프로세서에 할당된 경우를 비교해 작은 값을 선택하면 된다. 다음은 $ect(j)$의 계산식이다.

$$ect(j) = Min[(est(j) + EXC(j, fpred_1(j) \text{ 할당된 프로세서})), (est(j)_{Max} + EXC(j, sep_1(j)))] \quad (13)$$

$ect(j)$는 한 개의 부모 노드가 있는 경우의 식 (9)와 2개 이상의 부모 노드가 있는 경우의 $ect(j)$인 식 (13)을 비교해 보면 두 노드사이의 통신 시간(CMC)이 제외 된 것을 볼 수 있다. 이것은 $ect(j)$에 통신 시간이 포함되어 있기 때문이다. 한 개의 부모 노드가 있는 경우 통신 시간을 감소시키기 위해 부모 노드의 복제를 통해 자식 노드와 동일 프로세서에 할당 한다. 따라서 부모 노드와 자식 노드 사이에는 통신 시간은 0 이다. 그러나 2개 이상의 부모 노드가 존재하는 경우에는 $fpred_1(j)$인 부모 노드의 복제만으로 $est(j)$을 구할 수 없기 때문에 $fpred_1(j)$, $ect(j)$와 $est(j)_{Max}$을 구하는 식 (10), (11), (12)에 통신 시간을 포함 하였다.

3.1.2 단계 2 - 클러스터 구성

단계 2는 단계 1에서 계산된 est, ect, $fpred$, sep 값을 이용해 클러스터를 구성하는 부분이다. 우선 단계 1의 결과 값을 이용해 아래와 같이 표 1을 구성한다.

각 노드의 $fpred_1$을 이용해 클러스터를 구성한다. 클러스터 구성의 설명을 위해 그림 2의 입력 그래프에서 노드 13의 $fpred_1(13)=11$, $fpred_1(11)=10$, $fpred_1(10)=6$, $fpred_1(6)=2$, $fpred_1(2)=1$, $fpred_1(9)=6$, $fpred_2(9)=5$라 가정한다. 하향식으로 계산된 각 노드의 $fpred_1$ 정보를 이용해 상향식으로 클러스터를 구성해야 한다. 따라서 노드 13, 노드 12,... 순서로 클러스터를 구성해 나간다. 첫 번째 클러스터를 구성하기 위해 노드 13을 기준으로 시작하면 노드 13의 $fpred_1(13)=11$ 이므로 노드 13과 노드 11은 하나의 클러스터로 구성된다. 다음으로 노드 11의 $fpred_1(11)=10$ 이므로 다시 노드 13, 11, 10을 하나의 클러스터로 구성한다. 이와 같은 방법으로 마지막 노드의 $fpred_1$ 정보를 이용해 클러스터를 구성하면 13→11→10→6→2→1이 된다. 다음으로 노드 12를 기준으로 클러스터를 구성한다. 노드 12의 부모 노드는 1개만 존재하므로 $fpred_1(12)=8$이 된다. 노드 8 역시 1개의 부모 노드만 존재하므로 이를 이용해 두 번째 클러스터를 구성하면 12→8→4→1이 된다. 노드 1과 같이 노드 1의 자식 노드가 각각 1개의 부모 노드를 가지고 있는 경우는 노드 1의 복제를 통해 클러스터에 할당하게 된다. 다음으로 노드 11을 기준으로 세 번째 클러스터를 구성해야 한다. 하지만 노드 11과 노드 10은 이미 첫 번째 클러스터 구성 시 할당되었기 때문에 아직 할당되지 않은 노드 9를 기준으로 클러스터를 구성한다. 노드 9의 $fpred_1(9)=6$, $fpred_2(9)=5$이다. 따라서 세 번째 클러스터를 9→6으로 구성해야 하지만, 노드 6은 첫 번째 클러스터 구성 시 할당되었기 때문에 $fpred_2(9)=5$을 이용해 9→5로 클러스터를 구성한다. 노드 5 역시 한 개의 부모 노드를 가지고 있으므로 노드 2를 선택한다. 따라서 세 번째 클러스터는 9→5→2→1이 된다. 마지막으로 할당되지 않은 노드 7을 기준으로 위와 같은 방식으로 7→3→1인 클러스터를 구성할 수 있다. 이와 같이 각 노드의 $fpred_1$ 정보를 이용해 클러스터를 구성한다.

표 1. est, ect, $fpred$, sep 결과 테이블

$node$	est	ect	$fpred$			sep		
			$fpred_1$	$fpred_2$	⋯	sep_1	sep_2	⋯
1	0	2	0	0	⋯	p_1	p_3	⋯
2	2	7	1	0	⋯	p_2	p_1	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

3.1.3 단계 3 - 최적 프로세서 선택 및 할당

단계 2에서 우리는 노드간의 선후 관계를 이용해 클러스터를 구성했다. 다음 단계는 구성된 클러스터를 최적 프로세서에 할당 하는 것이다. 이것은 단순히 선택된 노드를 최적 프로세서에 할당 하는 것이 아니라 클러스터 전체가 최소의 수행 시간을 가질 수 있도록 최적의 프로세서를 찾아 할당해야 한다. 본 논문에서는 클러스터의 전체 수행 시간이 최소가 되기 위한 최적 프로세서를 선택할 수 있는 방법을 제시한다.

• 방법 1 : 클러스터의 최적 프로세서 선택

구성된 클러스터에 포함된 모든 노드의 수행 시간 합이 최소가 되는 프로세서를 찾는다. 첫 번째 클러스터의 모든 노드의 수행 시간 합이 프로세서 p에서 최소이면 첫 번째 클러스터는 프로세서 p을 선택한다. 두 번째 클러스터는 첫 번째 클러스터가 프로세서 p을 선택했기 때문에 프로세서 p을 제외하고 수행 시간의 합을 계산한다. 나머지 프로세서에서 수행 시간의 합이 최소가 되는 프로세서를 선

택한다. 만약 수행 시간의 합이 동일할 경우에는 클러스터의 구성 노드 중 처음 노드의 수행 시간이 최소가 되는 프로세서를 선택한다. 첫 번째 노드가 동일하면 순차적으로 두 번째, 세 번째 노드 순으로 비교 후 프로세서를 선택한다. 즉 해당 클러스터의 sep을 선택하는 것이다. 이때 해당 클러스터의 sep_1가 이미 선택되었다면 sep_2 프로세서를 선택한다.

• 방법 2 : 각 노드의 최적 프로세서 선택

방법 1에서 선택된 프로세서를 바탕으로 클러스터를 프로세서에 할당 한다. 이때 방법 1에서 선택된 프로세서를 기준으로 클러스터의 두 번째 노드에 대한 최적 프로세서를 선택한다. $k \in PRED(j)$, $fpred_1(j)=k$인 경우 노드 k와 j가 동일한 클러스터에 있고, 노드 k을 클러스터의 첫 번째 노드, 노드 j을 두 번째 노드, 프로세서 p을 방법 1에서 선택된 최적 프로세서라 가정한다. 첫 번째 노드 k는 방법 1에서 선택된 프로세서에 할당한다. 두 번째 노드 j을 프로세서 p에 할당하는 경우와 $sep_1(j)$에 할당하는 경우를 비교하여 최소의 $ect(j)$을 갖는 프로세서를 선택한다. 클러스터 내의 나머지 노드 역시 위의 방법을 이용해 프로세서를 선택한다. 노드 k와 j가 클러스터의 첫 번째, 두 번째 노드가 아닌 세 번째, 네 번째 노드인 경우 k의 프로세서는 방법 1에서 선택된 프로세서 p일수도 있고, $sep_1(k)$일 수도 있다. 즉 현재 할당 하려는 노드의 이전 노드가 어떻게 할당 되는가에 따라 프로세서가 결정된다. 노드 k와 노드 j가 동일 프로세서 할당될 조건은 식 (14)와 같다. 아래 식을 만족하면 노드 j 역시 노드 k의 프로세서에 할당된다.

$$ect(k)+EXC(j,p) \le (ect(k)+CMC(k,j) + EXC(j,sep_1(j))) \quad (14)$$

노드 k가 할당된 프로세서와는 다른 프로세서인 $sep_1(j)$에 노드 j가 할당될 조건은 식 (15)과 같다.

$$ect(k)+EXC(j,p) > (ect(k)+CMC(k,j) + EXC(j,sep_1(j))) \quad (15)$$

3.1.4 단계 4 - 메시지 스케줄링 및 태스크 복제

단계 3에서 모든 클러스터를 프로세서에 할당 되었다면 마지막으로 메시지 스케줄링을 수행한다. 메시지 스케줄링은 현재 노드가 자신의 모든 부모 노드로부터 데이터를 받을 수 있도록 수행하면 된다. 메시지 스케줄링 후 태스크 복제가 필요하면 다음의 두 가지 방법을 이용해 태스크 복제를 수행한다.

(1) 여분의 프로세서를 이용한 태스크 복제

사용가능한 프로세서를 m개, 실제 클러스터를 할당할 프로세서를 m'개, 노드 j의 $fpred_1(j)=k$, $fpred_2(j)=l$ 이라고 가정한다. $m > m'$ 이면 아직 사용하지 않은 프로세서 $m-m'$개가 존재한다. 즉 여분의 프로세서를 이용한 태스크 복제 방법은 $m-m'$개의 프로세서를 이용해 태스크를 복제하는 것이다. 클러스터 구성 시 노드 j의 $fpred_1(j)$을 이전 클러스터 구성 때 사용하였다면 $fpred_2(j)$을 이용해 클러스터를 구성해야 한다. 하지만 최소의 수행 시간을 얻기 위해서는 노드 j와 $fpred_1(j)=k$을 동일 프로세서에 할당하여 통신 비용 $CMC(k,j)$을 제거하여야 한다. 따라서 여분의 프로세서를 이용한 태스크 복제는 노드 j와 $fpred_1(j)=k$가 동일 프로세서에 할당될 수 있도록 여분의 프로세서에 두 노드의 복제를 통해 새로운 클러스터를 생성한다. 이것은 프로세서를 사용하는 수는 증가하지만 전체 수행 시간을 감소시킬 수 있다. 하지만 프로세서 마다 노드의 수행 시간이 다르므로 오히려 더 늦어질 경우가 발생하기 때문에 언제나 복제를 수행하거나, 수행 시간의 감소를 보장하지 못한다.

(2) 빈 슬롯을 이용한 태스크 복제

노드 j의 $\{k,l\} \subset PRED(j)$, $fpred_1(j)=k$, $fpred_2(j)=l$, 노드 j와 노드 l은 동일 클러스터에 위치하고 프로세서 p에 할당되어 있으며 $fpred_1(j)$인 노드 k는 다른 클러스터에 이미 할당되어 있다고 가정한다. 이때 빈 슬롯을 이용한 태스크 복제 방법은 동일 클러스터 내에 있는 노드 l과 노드 j 사이에 메시지 스케줄링 이후 발생하는 빈 슬롯을 이용해 $fpred_1(j)=k$ 노드를 복제하는 방법이다. 물론 노드 l과 노드 j 사이의 빈 슬롯에 노드 k가 복제될 수 있는 시간적 공간이 존재해야 한다. 즉, 노드 l과 노드 j 사이의 빈 슬롯의 공간을 s 라고 가정하면, $s > EXC(k/p)$인 경우 복제가 가능하며 $s-EXC(k/p)$ 만큼의 시간적 감소를 가져올 수 있다.

3.2 입력 그래프에 따른 TANH과 DTSC 알고리즘의 성능 비교

선형 입력 그래프와 일반적인 입력 그래프를 바

탕으로 TANH 스케줄링 기법과 본 논문에서 제안된 DTSC 스케줄링의 수행 과정과 두 알고리즘의 결과 길이를 비교한다.

3.2.1 선형 입력 그래프

시작 노드 1, 마지막 노드 3, 중간 노드 2의 3개의 노드를 가지는 선형 입력 그래프를 가정한다. 선형 입력 그래프는 각 노드의 부모노드가 하나인 노드이다. 이때 통신 시간은 모두 1이라고 가정하고 각 노드가 p_1, p_2, p_3 프로세서에서 실행 될 때의 수행 시간을 표 2와 같이 가정 하였다. 이 때 TANH 알고리즘과 본 논문에서 제안한 DTSC 알고리즘에 적용하여 알고리즘의 동작 방법과 결과를 구하면 다음과 같다.

표 2. 노드의 수행 시간

node	p_1	p_2	p_3
1	10	100	100
2	100	10	100
3	100	100	10

(1) TANH 스케줄링 결과

각 노드는 오직 한 개의 부모 노드만 존재하므로 $fpred_1(3)=2$, $fpred_2(2)=1$이고 $fpred_1(1)$은 입력 노드이므로 존재하지 않는다. 따라서 클러스터는 3→2→1로 구성된다. TANH 스케줄링 방법에 의해 클러스터의 헤더 노드인 노드 3은 $fproc_1(3)=p_3$에 할당된다. 한 개의 클러스터만 존재하므로 메시지 스케줄링은 필요하지 않으며 스케줄링 결과는 그림 3(a)와 같다.

(2) DTSC 스케줄링 결과

각 노드는 오직 1개의 부모 노드를 가지고 있으므로 1개의 클러스터로 구성됨을 알 수 있다. 입력 그래프를 이용해 DTSC 스케줄링을 수행하면 다음과 같다.

• 단계 1 - *est*, *ect*, *fpred*, *sep* 계산

각 노드의 *est*, *ect*, *fpred*, *sep*는 본 논문에서 제안한 식 (6), (7), (8), (9)을 이용하여 구할 수 있으며 계산된 결과값은 표 3과 같다.

• 단계 2 - 클러스터 구성

각 노드는 오직 한 개의 부모 노드만 존재하므로 $fpred_1(3)=2$, $fpred_1(2)=1$이고 $fpred_1(1)$은 입력 노드이므로 존재하지 않는다. 따라서 클러스터는 3→2→1로 구성된다.

• 단계 3 - 최적 프로세서 할당 및 선택

방법 1을 이용해 클러스터 3→2→1의 수행 시간 합이 최소가 되는 프로세서를 찾으면 모든 프로세서에서 수행 시간 합이 210으로 동일한 결과가 나온다. 따라서 클러스터의 첫 번째 노드 1의 수행 시간이 최소가 되는 프로세서 p_1을 선택한다. 방법 1에서 선택된 프로세서 p_1을 기준으로 방법 2를 적용하면 다음과 같다.

노드1과 2 : $ect(1)+EXC(2,p_1)$

$> ect(1)+CMC(1,2)+EXC(2,p_1)$

노드2과 3 : $ect(2)+EXC(3,p_2)$

$> ect(2)+CMC(2,3)+EXC(3,p_3)$

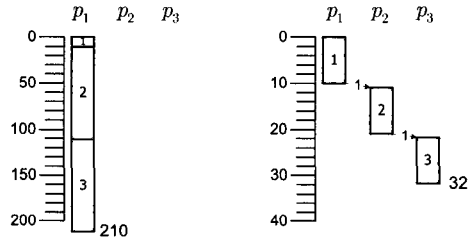
따라서 노드 1과 2에서 노드 2의 프로세서는 방법 2의 식 (15)을 만족하므로 노드 2의 $sep_1(2)=p_2$에 할당하게 된다. 노드 2와 3에서 노드 3의 프로세서는 역시 방법 2의 식 (15)을 만족하므로 노드 3의 $sep_1(3)=p_3$에 할당하게 된다.

표 3. *est*, *ect*, *fpred*, *sep* 결과 값

node	*est*	*ect*	*fpred*	*sep*		
				sep_1	sep_2	sep_3
1	0	10	0	p_1	p_2	p_3
2	10	21	1	p_2	p_1	p_3
3	21	32	2	p_3	p_1	p_2

• 단계 4 - 메시지 스케줄링 및 태스크 복제

1개의 클러스터만이 존재하므로 메시지 스케줄링을 수행 할 필요가 없다. 따라서 최종 스케줄링 결과는 그림 3 (b)와 같으며 총 결과 길이는 32이가 되어 TANH 알고리즘 보다 월등한 성능향상을 확인 할 수 있다.



(a) TANH 결과 길이 210 (b) DTSC 결과 길이 32

그림 3. TANH와 DTSC 결과 길이.

3.2.2 일반적인 입력 그래프 경우

일반적인 입력 그래프와 프로세서에 따른 노드의 수행 시간을 그림4와 같이 가정하였다. 입력 그래프에서 모든 노드간의 통신 시간을 2로 가정하고 그림 4의 노드 수행 시간표에서 굵게 표시한 것은 해당 노드의 가장 짧은 수행 시간을 표시한 것이다. 따라서 이 값을 갖는 프로세서가 해당 노드의 sep_1 된다.

node	p_1	p_2	p_3	p_4	p_5	p_6
1	3	3	4	2	5	4
2	5	6	6	5	5	4
3	2	3	4	2	4	3
4	5	3	5	3	5	3
5	7	7	6	5	5	6
6	3	4	2	3	2	4
7	6	7	7	5	7	5
8	5	7	5	6	6	5
9	7	7	5	6	6	5
10	2	3	2	2	3	2
11	3	2	3	3	2	3

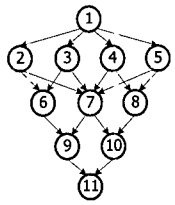


그림 4. 일반적인 입력 그래프와 노드 수행 시간.

(1) TANH 스케줄링 결과

TANH 스케줄링은 총 4 단계로 구성되어 있다. 그림 4의 입력 그래프를 이용해 각 단계별로 *est*, *ect*, *fpred*, *fproc*, *last*, *lact*, *level*을 계산하면 표 4의 결과를 얻을 수 있다[10]. 레벨 값을 이용해서 오름차순으로 노드를 정렬하면 {11, 10, 9, 8, 6, 7, 3, 4, 2, 5, 1}이 되고 각 노드의 *fpred* 정보를 이용하여 클러스터를 구성한다. 첫 번째 클러스터는 정렬된 결과의 첫 번째 노드인 11을 기준으로 구성해 나간다. 노드 11의 $fpred(11)=9$ 이므로 11→ 9가 되고 노드 9의 $fpred(9)=7$, 노드 7의 $fpred(7)=2$, 노드 2의 $fpred(2)=1$ 이므로 클러스터를 구성하면 11→9→7→2→1 이 된다. 두 번째 노드는 정렬된 결과에서 아직 할당 하지 않은 노드를 기준으로 구성한다. 따라서 노드 10을 기준으로 다음 클러스터를 구성하면 $fpred(10)=7$ 이지만 노드 7은 이미 할당되었기 때문에 노드 10의 부모 노드 중 아직 할당되지 않은 노드 8을 선택한다. 노드 8의 $fpred(8)=5$, $fpred(5)=1$ 이므로 두 번째 클러스터를 구성하면 10→8→5→1이 된다. 이와 같이 나머지 모든 노드를 이용해 클러스터를 구성하면 6→3→1, 4→1을 구성할 수 있다.

각 클러스터를 할당하기 위한 프로세서는 클러스터의 첫 번째 노드인 노드 11과 노드 10, 노드 6, 노드 4의 $fproc_1$인 프로세서이다 첫 번째 클러스터

표 4. TANH의 결과값($fp_i = fproc$)

node	level	est	fp_1	fp_2	fp_3	fp_4	fp_5	fp_6	ect	fpred	last	lact
1	29	0	4	1	2	3	6	5	2	-	0	2
2	23	2	4	6	1	5	2	3	7	1	4	9
3	21	2	4	1	2	6	3	5	4	1	5	7
4	22	2	4	2	6	1	3	5	5	1	2	5
5	24	2	4	5	3	6	1	2	7	1	2	7
6	14	7	4	3	5	1	2	6	10	2	9	12
7	17	9	6	4	1	2	3	5	14	2	9	14
8	13	7	4	1	3	6	5	2	13	5	7	13
9	10	14	6	3	5	4	1	2	19	7	14	19
10	6	15	6	1	3	4	2	5	17	7	15	17
11	3	19	6	2	5	1	3	4	22	9	19	22

가 $fproc_1(11)=6$을 선택했으므로 두 번째 클러스터는 $fproc_1(10)=6$이 아닌 $fproc_2(10)=1$인 프로세서 1을 선택한다. 각 클러스터의 최적 프로세서를 선택한 후 메시지 스케줄링을 수행하면 그림 5(a)처럼 5개의 프로세서를 사용함으로써 25의 결과 길이를 얻을 수 있다.

(2) DTSC 스케줄링 결과

입력 그래프를 이용해 DTSC 스케줄링을 수행하면 다음과 같다.

• 단계 1 - *est*, *ect*, *fpred*, *sep* 계산

각 노드의 *est*, *ect*, *fpred*, *sep*는 본 논문에서 제안한 식 (6),(7),(8),(9)을 이용하여 구할 수 있으며 계산된 결과 값을 정리하면 표 5와 같다.

• 단계 2 - 클러스터 구성

마지막 노드 11부터 클러스터를 구성하면 다음과 같다. 노드 11의 $fpred_1(11)=9$이므로 노드 9를 선택한다. 노드 9의 $fpred_1(9)=7$, 이런 식으로 첫 번째 클러스터를 구성하면 11→9→7→5→1이 된다. 두 번째 클러스터는 아직 할당되지 않은 노드 10을 기준으로 구성한다. 노드 10의 $fpred_1(10)=7$, $fpred_2(10)=8$이다. 하지만 노드 7은 이미 할당되었기 때문에 $fpred_2(10)=8$인 노드 8을 선택한다. 따라서 두 번째 클러스터는 10→8→4→1이 된다. 세 번째 클러스터는 노드 6을 기준으로 구성한다. 노드 6의 $fpred_1(6)=2$ 이므로 6→2→1로 클러스터가 구성된다. 마지막으로 아직 할당되지 않은 노드 3을 기준으로 3→1의 클러스터를 구성한다.

표 5. *est*, *ect*, *fpred*, *sep* 결과값

node	*est*	*ect*	*fpred*(= *fr*)				*sep*					
			fr_1	fr_2	fr_3	fr_4	sep_1	sep_2	sep_3	sep_4	sep_5	sep_6
1	0	2	0	0	0	0	p_4	p_1	p_2	p_3	p_6	p_5
2	2	7	1	0	0	0	p_6	p_1	p_4	p_5	p_2	p_4
3	2	4	1	0	0	0	p_1	p_1	p_2	p_6	p_3	p_5
4	2	5	1	0	0	0	p_2	p_4	p_5	p_1	p_3	p_5
5	2	7	1	0	0	0	p_4	p_5	p_4	p_6	p_1	p_2
6	7	11	2	3	0	0	p_3	p_5	p_1	p_4	p_2	p_6
7	9	14	5	2	4	3	p_4	p_6	p_1	p_2	p_3	p_5
8	7	13	5	4	0	0	p_1	p_3	p_6	p_4	p_5	p_2
9	14	20	7	6	0	0	p_3	p_6	p_4	p_5	p_1	p_2
10	15	17	7	8	0	0	p_1	p_3	p_4	p_6	p_2	p_5
11	20	23	9	10	0	0	p_2	p_5	p_4	p_3	p_4	p_6

• 단계 3 - 최적 프로세서 할당 및 선택

방법 1을 이용해 클러스터의 전체 수행 시간 합이 최소가 되는 프로세서를 찾는다. 방법 1에서 선택된 프로세서 p_1을 기준으로 방법 2를 적용한다. 방법 2의 적용 방법은 다음과 같다.

노드1과5 : $ect(1)+CPC(5,p_4) \leqq ect(1)+CMC(1,5)+CPC(5,p_4)$

:

노드9와11 : $ect(9)+CPC(11,p_4) \leqq ect(9)+CMC(9,11)+CPC(11,p_2)$

노드1과4 : $ect(1)+CPC(4,p_6) \leqq ect(1)+CMC(1,4)+CPC(4,p_2)$

:

노드8과10 : $ect(8)+CPC(10,p_6) \leqq ect(8)+CMC(8,10)+CPC(10,p_{10})$

노드1과2 : $ect(1)+CPC(2,p_1) \leqq ect(1)+CMC(1,2)+CPC(2,p_6)$

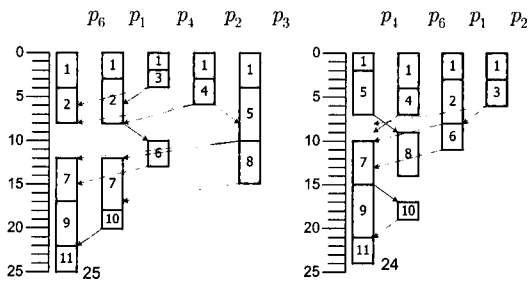
노드2와6 : $ect(2)+CPC(6,p_1) \leqq ect(2)+CMC(2,6)+CPC(6,p_3)$

노드1과3 : $ect(1)+CPC(3,p_2) \leqq ect(1)+CMC(1,3)+CPC(3,p_1)$

모든 결과가 방법 2의 식 (14)을 만족하므로 방법 1에서 선택한 클러스터의 최적 프로세서에 할당하면 된다.

• 단계 4 - 메시지 스케줄링 및 태스크 복제

단계 4에서 선택된 프로세서에 클러스터를 할당 후 노드간의 선후 관계를 이용해 메시지 스케줄링을 수행하면 스케줄링이 완료된다.



(a) TANH 결과 길이 25 (b) DTSC 결과 길이 24

그림 5. TANH와 DTSC 결과 길이.

그림 5(b)와 같이 4개의 프로세서를 사용해서 24의 결과 길이 얻었다. 그림 5(b)에서 여분의 프로세서를 이용한 태스크 복제 방법과 빈 슬롯을 이용한 태스크 복제는 사용되지 않았다. 각 노드의 $fpred_1$을 복제할만한 빈 슬롯이 발생하지 않았기 때문이다. 이것은 통신 시간이 수행 시간보다 작은 경우가 많기 때문이다. 일반적인 입력 그래프를 이용하였을 경우 TANH 스케줄링 방법은 5개의 프로세서를 사용해 결과 길이 25를 얻었고, 제안된 DTSC 스케줄링 방법은 4개의 프로세서를 사용해 결과 길이 24를 얻었다.

Ⅳ. 시뮬레이션 및 성능분석

4.1 시뮬레이션 환경

이번 장에서는 제안된 태스크 스케줄링 알고리즘의 성능을 분석 한다. DTSC 스케줄링의 성능을 측정하기 위해서 다양한 개수의 노드와 이질 시스템 환경을 반영하기 위해 다양한 수행 시간을 갖는 프로세서에 대해 병렬성을 가지는 입력 그래프를 무작위로 만들었다. 즉 30, 50, 100, 150, 200, 250, 300개의 노드를 2, 3, 5, 7, 10, 12, 15, 20, 25, 30, 35, 40, 45, 50개의 프로세서 상에 스케줄링 하였다. 여기서 각 노드와 프로세서의 쌍에 대해 서로 다른 무작위 그래프를 만들어 성능을 비교 분석하였다.

이질 시스템 환경을 반영하기 위해 태스크의 수행 시간은 기본 수행 시간 값의 ±80% 범위 내에서

무작위로 설정 하였다. 기본 수행 시간을 5라 하면 최대 9의 수행 시간과 최소 1의 수행 시간을 가질 수 있다. 또한 통신 시간에서도 수행 시간과 같은 방식으로 기본 통신 시간 값의 ±80% 범위 내에서 무작위로 설정 하였다. 통신 시간의 변화 범위를 지나치게 크게 할 경우 태스크 스케줄링의 과정에서 큰 통신 시간 값에 의해 잘못된 스케줄링 결과를 발생 하게 된다. 따라서 본 논문의 시뮬레이션 과정에서 이런 부분을 제외하기 위해 변화 폭을 제한하기로 한다. 또한 수행 시간과 통신 시간의 기본 시간들은 변화 폭을 포함하여 서로 중첩 되게 설정한다. 이것은 통신 시간이 수행 시간 보다 큰 경우와 수행 시간이 통신 시간보다 큰 경우를 나타내기 위함이다. 통신 시간이 수행 시간 보다 큰 경우 클러스터를 할당한 프로세서에 다른 프로세서를 복제하거나 할당할 수 있는 공간이 발생 한다는 의미이다. 이런 경우 프로세서의 빈 공간을 이용해 노드의 복제를 통해 좀 더 효율적인 스케줄링을 수행 할 수 있다. 반면 통신 시간이 수행 시간 보다 작다면 위의 경우와 같은 상황은 발생 하지 않는다.

표 6. 시뮬레이션 파라미터

파라미터	설정 값(범위)	비고
노드 개수	30, 50, 100, 150, 200, 250, 300	
프로세서 개수	2, 3, 5, 7, 10, 12, 15, 20, 25, 30, 35, 40, 45, 50	
기본 통신시간	5	±80%
기본 수행시간	5	±80%
DAG 개수	5	

스케줄링의 성능 비교는 현존하는 클러스터 스케줄링인 TANH, GDL 그리고 제안된 DTSC 알고리즘에 대해서 수행한다.

4.2 성능 비교 기준

스케줄링 성능 비교를 위해 항목별로 비교한다. 각 항목에 대해 무작위로 만들어진 입력그래프에 따른 스케줄링의 성능을 비교한다.

• 스케줄링 결과 길이 비

스케줄링의 결과 길이 비는 입력 그래프에 대해 실제 스케줄링 결과 값으로 이상적인 *ect*의 값을 나눈 것으로 정의 한다. 여기서 이상적인 *ect*값은 제안한 식에 의해 계산된 값을 말한다.

$$\text{스케줄링 결과 길이비} = \frac{\text{이상적인 } ect \text{ 값}}{\text{실제 스케줄링 결과 길이}}$$

스케줄링 결과 길이 비는 스케줄링의 정확성을 의미한다. 주어진 *est*, *ect*, *fpred*, *sep*의 값을 이용해 이상적인 *ect* 값이 실제 스케줄링의 결과 길이와 차가 크면 현실성이 없는 이론적인 스케줄링 알고리즘에 불가 하다. 따라서 스케줄링 결과 길이 비가 1에 가까울수록 정확한 스케줄링을 수행 한다는 것을 알 수 있다.

• CCR 비교

CCR은 통신 시간과 수행 시간의 비이다. 앞에서 언급했듯이 통신 시간이 수행 시간 보다 클 경우 프로세서의 빈 공간에 노드의 복제가 발생할 수 있다. CCR은 다음과 같이 정의 한다.

$$CCR = \frac{\text{평균 통신 시간}}{\text{평균 노드의 수행시간}}$$

• 프로세서 사용 빈도

프로세서의 사용 빈도는 매 시간 마다 현재 프로세서가 사용되고 있는지, 다시 말해서 태스크를 수행하고 있는지를 나타낸다. 프로세서 사용 빈도는 어떤 시점에 얼마만큼의 프로세서를 사용하고 있는지를 보여준다. 이 값은 시뮬레이션 수행 과정에서 각 타임 마다 프로세서의 사용 개수를 저장하여 비교 하면 알 수 있다. 또한 시간당 프로세서의 개수가 적다는 것은 전체적으로 적은 프로세서를 사용해 스케줄링을 수행 한다는 의미이다.

• 프로세서 완료 시간

사용된 프로세서 완료 시간이란 프로세서가 작업을 마치고 다른 태스크를 처리할 수 있는 상태를 말한다. 즉, 프로세서의 완료 시간이 빠를수록 입력 그래프에 대한 작업이 빨리 끝난 것을 의미하고 새롭게 다른 작업을 할당할 수 있다는 것이다. 완료 시간은 메시지 스케줄링 이후 사용된 프로세서의 마지막으로 할당된 노드의 실제 *ect*을 보면 알 수 있다.

4.3 시뮬레이션 결과

시뮬레이션 환경 설정에서 언급했듯이 30, 50, 100, 150, 200, 250, 300개의 노드를 2, 3, 5, 7, 10, 12, 15, 20, 25, 30, 35, 40, 45, 50개의 프로세서 상에 성능 비교 기준(4.2 성능 비교 기준)을 바탕으로 스케줄링을 수행 한다.

그림 6은 스케줄링 결과 길이 비이다. 스케줄링

결과 길이 비의 정의를 보면 이상적인 *ect*을 실제 결과 길이로 나눈 것이다. 따라서 1보다 작다는 것은 실제 스케줄링 결과는 가장 빨리 완료 할 수 있는 시간 *est* 보다 크다는 것을 알 수 있다. 마지막 단계인 메시지 스케줄링 상에서 나타나는 결과이다.

또한 전체적으로 노드가 증가 할수록 스케줄링 결과 길이 비는 감소하고 있다. 이것은 노드가 증가할수록 부모 노드와 자식 노드간의 선후 관계에 의해 각 노드의 계산된 *ect* 보다 늦추어 진다는 것을 보여준다. 또한 노드의 증가에 따른 곡선의 기울기가 완만한 것은 이상적인 *ect* 값이 실제 스케줄링 결과와 유사함을 나타낸다. 전체적으로 DTSC가 좋은 성능을 보여줌을 알 수 있다.

그림 7은 노드 개수에 따른 'TANH 결과 길이/DTSC 결과 길이' 이다. 이것은 두 알고리즘의 성능 비교를 나타낸 것이다. 결과 길이가 짧을수록 좋은 스케줄링을 수행 했다는 의미이다. 따라서 'TANH 결과 길이/DTSC 결과 길이' 값이 1보다 작다는 것은 'DTSC 결과 길이가 TANH 결과 길이' 보다 작다는 의미이고 이것은 좋은 스케줄링을 수행 했다는 의미이다. 노드가 50개 있고, $CCR=0.2$일 때 1보다 작은 값을 보여 주고 있다. 이것은 $CCR=0.2$이므로 노드의 수행 시간이 통신 시간보다 큰 경우가 많기 때문에 노드의 복제를 수행 하지 못했기 때문이다. 노드가 증가 할수록 *CCR*에 관계없이 DTSC가 TANH보다 좋은 결과를 보여 준다.

프로세서의 사용 빈도는 수행 시간에 따라 사용중인 프로세서의 개수를 의미 한다. 그림 8은 최대 노드 개수 300개, 최대 프로세서 개수 50을 기준으로 수행된 결과이다. 300개의 노드가 있는 DAG에서는 스케줄링간의 사용 중인 프로세서의 개수는 큰 차이가 나타나지 않는다. 스케줄링이 완료 되면

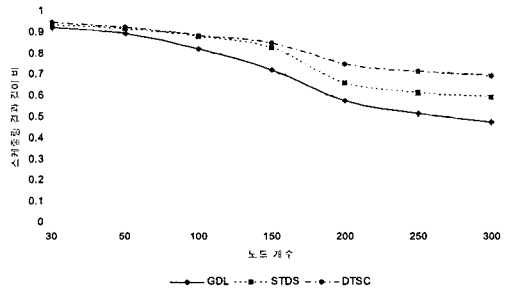


그림 6. 스케줄링 결과 길이 비교

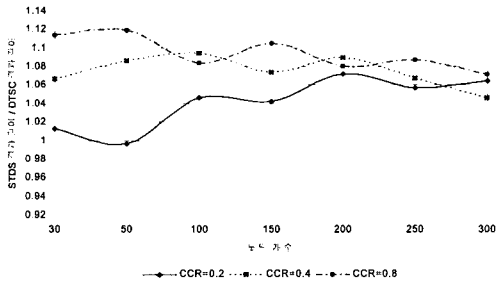


그림 7. (TANH 결과 길이)/(DTSC 결과 길이)의 변화

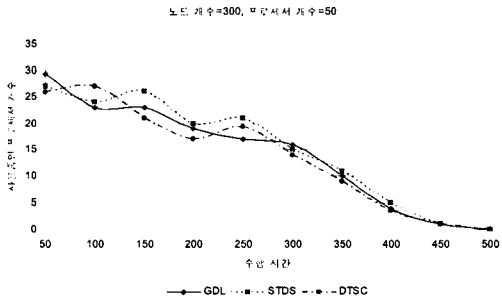


그림 8. 프로세서 사용 개수

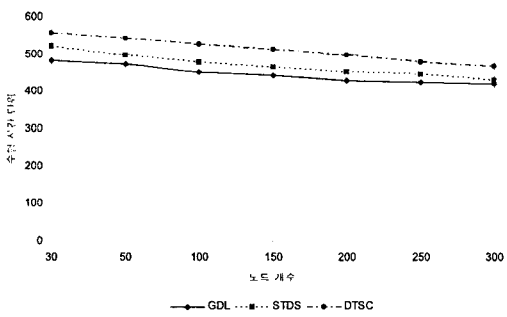


그림 9. 프로세서 완료 시간 합

서 클러스터가 할당되었던 프로세서 혹은 노드가 할당되었던 프로세서에서 노드의 수행이 완료되면서 프로세서가 이상적인 상태로 돌아가기 때문에 우측 하향 곡선의 형태를 나태내고 있는 것이다. 수행 시간이 늘어날수록 사용 중인 프로세서가 선형적으로 감소되어야 하나 프로세서에 할당된 노드의 선후 관계에 인한 통신 시간 때문에 프로세서에 빈 슬롯이 발행하기 때문이다.

프로세서의 완료 시간이 적을수록 프로세서의 이상적인 상태가 많아지므로 다른 작업을 수행할 수 있다. 그림 21은 각 프로세서의 완료 시간의 합을 나타내고 있다. 노드의 개수가 증가할수록 완료 시

간의 합은 감소하며 DTSC가 가장 좋은 결과를 보여주고 있다. 클러스터 구성 시 통신 시간을 감소시키기 위해 $fpred_1$인 노드를 같은 프로세서 할당하기 때문에 프로세서의 이용률이 증가하고 결과적으로 프로세서 내에서 노드와 노드 사이의 빈 슬롯이 발생하지 않기 때문에 완료 시간의 합이 증가하게 된다.

V. 결 론

본 논문은 이질 시스템에서 프로세서간의 통신 시간 및 노드들 간의 선후 관계를 충분히 고려한 DTSC 스케줄링을 제안하였고, 시뮬레이션을 통해 현존하는 태스크 스케줄링 알고리즘과 성능을 비교했다. 여러 개의 부모 노드가 존재 할 경우 부모 노드의 다양한 경우를 수식적으로 표현했으며, 클러스터를 할당하기 위한 방법으로 프로세서 선택 시 최소의 결과 길이를 얻을 수 있는 2가지 방법을 제안하였다.

제안된 스케줄링 방법은 입력 그래프가 선형적인 경우는 클러스터를 프로세서에 할당하는 단계 3의 방법 2에 의해 최적의 스케줄링 결과를 보여주었으며, 일반적인 무작위 입력 그래프 경우에서도 좋은 결과를 보여 주었다. 무작위 입력 그래프인 경우 스케줄링 결과 길이비의 시뮬레이션 결과를 통해 우리는 DTSC가 다른 알고리즘에 비해 더욱 정확한 est을 계산하는 것을 알 수 있다. 또한 스케줄링 결과 길이에서도 좋은 결과를 보여 주었다. 결과적으로 무작위 입력 그래프에 대해 DTSC 스케줄링은 TANH 스케줄링에 비해 2%, GDL 스케줄링에 비해 5%의 성능 향상을 보였나 특정 입력 그래프와 노드의 수가 많을 수록 DTSC 스케줄링의 성능 향상이 높아지는 것을 확인 하였다. 이는 본 논문에서 제안한 DTSC 알고리즘이 기존의 알고리즘과 비교하여 모든 입력에 대한 성능 향상과 더불어 특정 입력 그래프와 노드의 수가 많을수록 효율적인 복제 기반 알고리즘이라는 결과를 얻을 수 있다.

참 고 문 헌

[1] R. L. Graham, L. E. Lawler, J. K. Lenstra, and A. H. Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey," *Annals of Discrete Math.*, pp. 287-326, 1979.

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

[3] T. Cassavant and J. A. Kuhl, "Taxonomy of Scheduling in General Purpose Distributed Memory Systems," *IEEE Trans. Software Eng.*,vol. 14, no. 2, pp. 141-154, 1988.

[4] Haluk Topcuoglu, Salim Hariri, and Min-You Wu, "Performance-Effective and Low-complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 2, March 2002.

[5] Savina Bansal, Padam Kumar, and Kuldip Singh, "An Improved Duplication Strategy for Scheduling Precedence Constrained Graphs in Multiprocessor Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, Feb 2004.

[6] M. Wu and D. Gajski, "Hypertool : A programming Aid for Message Passing Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 330-343, July 1990.

[7] Y. Kwok and I. Ahman, "Dynamic Critical-Path Scheduling: An Effective Technique for Allocation Task Graphs to Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 5, pp. 506-521, May 1996.

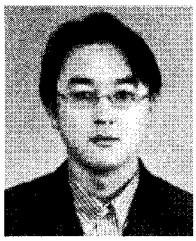
[8] T. Yang and A. Gerasoulis, "DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 9, pp. 951-967, Sept 1994.

[9] A. Ranaweera and D. P. Agrawal, "A Scalable Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *Proc. Int'l Conf. Parallel Processing*, pp. 383-390, Aug. 2000.

[10] Rashmi Bajaj and Dharma P. Agrawal, "Improving Scheduling of Tasks in a Heterogeneous Environment," *IEEE Trans. Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107-118, Feb. 2004.

[11] C. I. Park and T. Y. Choe, "An Optimal Scheduling Algorithm Based on Task Duplication," *IEEE Trans. Computers*, vol. 51, no. 4, pp. 444-448, Apr. 2002.

윤 완 오 (Wan-Oh Yoon) 준회원



2000년 2월 경기대학교 전자공학과 졸업
2002년 2월 인하대하교 전자공학과 석사
2002년 3월~현재 인하대학교 전자공학과 박사과정
<관심분야> 분산 처리 시스템, 병렬프로그래밍, 컴퓨터 아키텍쳐, 임베디드 시스템

백 정 규 (Jueng-Kuy Baek) 준회원

2003년 2월 인하대학교 전자공학과 졸업
2005년 2월 인하대학교 전자공학과 석사
2005년 3월~현재 LG전자(주)
<관심분야> 분산 처리 시스템, 병렬프로그래밍, 컴퓨터 아키텍쳐

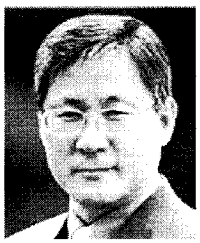
신 광 식 (Kwang-Sik Shin) 준회원

2001년 2월 인하대학교 전자공학과 졸업
2003년 2월 인하대학교 전자공학과 석사
2003년 3월~현재 인하대학교 전자공학과 박사과정
<관심분야> 멀티미디어 데이터 통신, 컴퓨터 네트워크, 무선 통신, 시스템 프로그래밍

정 진 하 (Jin-Ha Cheong) 준회원

1992년 2월 인하대학교 전자공학과 졸업
1994년 2월 인하대학교 전자공학과 석사
1994년~1999년 한미기술연구소(주)
2000년 3월~현재 인하대학교 전자공학과 박사과정
<관심분야> 컴퓨터 구조, 병렬 및 분산 처리시스템, 시스템 디자인

최 상 방 (Sang-Bang Choi) 종신회원



1981년 2월 한양대학교 전자공학과 졸업
1981년~1986년 LG정보통신(주)
1988년 3월 Univ. of Washington 석사
1990년 8월 Univ. of Washington 박사
1991년~현재 인하대학교 전자공학과 교수
<관심분야> 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템, Fault-tolerant computing