태스크 그래프의 재구성에 의한 효율적 태스크 스케줄링에 관한 연구

변 승 환† · 유 관 종††

요 약

본 논문은 병렬 처리 시스템 환경에서 효율적인 태스크 스케줄링에 관한 연구로써 태스크 그래프의 재구성에 의해 전체 수행 시간을 단축시키는데 목적을 두고 있다. 태스크 스케줄링은 m개의 태스크를 n개의 프로세서에 할당하는 연구인데 이는 많은 문제점을 갖고 있다.[1, 4, 9] 일반적으로 이 문제를 해결하는 것은 NP-hard 문제로 알려져 있다. 이러한 문제를 해결하고자 본 논문에서는 주어진 태스크 그래프를 재구성하여 스케줄링 하는 방법을 제시하였다. 태스크 그래프와 시스템 그래프를 이용하여 효과적으로 수행이 될 수 있는 재구성 태스크 그래프(RTG)를 만들고 이를 스케줄링 함으로써 기존의 논문에서 준 최적의 결과를 얻기 위해 태스크 스케줄링 후에 재할당 및 반복 수행의 과정이 사용하였는데 이를 없애면서 빠른 시간 안에 스케줄링이 이루어지도록 하였고 스케줄링의 결과 또한 향상시켰다.

A Study on the Efficient Task Scheduling by the Reconstructed Task Graph

Seung Hwan Byun† · Kwan Jong Yoo††

ABSTRACT

This paper presents an effective heuristic task scheduling algorithm for multiprocessor systems. To execute task scheduling effectively which is defined as an allocation of m's tasks onto n's processors(m > n), several problems almost at NP-hard should be cleaned up. The purpose of the task scheduling obtains the minimum execution time by mapping the tasks on a system topology or reduces the total execution time to give a minimum system topology. In order to solve this problem, in this paper, the task scheduling is done by redefining a task graph to a reconstructed task graph(RTG). An RTG is obtained by merging or copying nodes to equal the number of nodes on each level of the task graph to the number of processors of the system topology and then directly scheduled to the system topology. This method obtains a fast scheduling time and a simple scheduling method, and near-optimal execution time without executing steps such as the refinement step and the duplication step after the task scheduling.

† 정 회 원:서남대학교 전산정보학과 전임강사
†† 정 회 원:충남대학교 컴퓨터과학과 교수
논문접수:1997년 2월 24일, 심사완료:1997년 7월 28일

1. 서 론

병렬 처리 시스템에서 사용자가 작성한 순차 프로그램을 여러 개의 프로세서에서 병렬 수행될 수 있도록, 사용자의 순차 프로그램을 분할하여, 여러 프로세서에 분배하여 병렬 수행시킴으로써 최대의 효율성을 얻고자 하는 연구가 활발히 진행되고 있다[1, 2]. 병렬 처리와 관련된 연구의 목적은 대부분 프로그램 수행시 전체 수행 시간(TET, Total Execution Time)을 줄이는 것이므로, 순차 프로그램의 분할과 분할된 순차 프로그램의 할당에 대한 연구는 모두 전체 수행 시간의 단축에 목적을 두고 있다고 할 수 있다.

본 논문은 순차 프로그램을 병렬 처리 시스템에서 수행시키기 위해 순차 프로그램이 적당한 크기로 분할되고, 분할된 프로그램들의 종속성이 제시된 상황에서 전체 수행 시간을 줄일 수 있는 스케줄링 알고리즘의 개발에 목적을 두고 있다. 이 때, 본 논문에서 목표로 하는 스케줄링 알고리즘의 특징은 다음과 같다.

1) 간단하고 빠르게 스케줄링 결과를 제시해 주는 스케줄링 알고리즘.
2) 병렬 수행하고자 하는 프로그램의 가장 적정한 프로세서 수를 알 수 있도록 한다.
3) 프로그램 수행을 위한 가장 적당한 시스템 토폴로지(system topology)를 알 수 있도 록 한다.
4) 태스크 스케줄링 후 다시 재 할당하는 기법을 사용하지 않도록 한다.

먼저, 본 논문에서는 입력으로 제공된 태스크 그래프가 주어지면, 이 태스크 그래프의 수행 시간을 계산하기 위해 필요한 병렬 처리 시스템의 프로세서 수와 구성에 대한 정보를 얻는다. 이에 대한 정보는 주어진 태스크 그래프를 수행하기 위한 병렬 처리 시스템이 이미 존재하는 경우와 존재하지 않는 경우로 나눌 수 있다. 전자는 해당 시스템에 대한 정보를 이용하여 직접 본 논문에서 제시한 태스크 재구성 알고리즘과 재구성 된 태스크 그래프를 스케줄링 하는 스케줄러를 이용하여 전체 수행 시간을 계산한다. 그리고, 후자는 태스크 그래프를 수행시키고자 하는 병렬 처리 시스템이 존재하지 않는 경우이다. 따라서, 해당 태스크 그래프를 최소한의 수행 시간을 얻을 수 있는 시스템 환경(프로세서의 수, 프로세서의 구성)을 찾을 수 있도록 여러 병렬 처리 시스템 환경을 정보로 하여 전자와 같은 스케줄링 과정을 취한다. 본 논문에서는 태스크 그래프를 재구성한 후 이를 스케줄링 한 스케줄링 결과를 간트 차트 형태로 나타내었다.

본 연구의 목적은 효율적인 스케줄링이 이루어질 수 있도록, 최초의 태스크 그래프로부터 태스크 그래프 노드의 특성과 태스크 그래프를 수행시키기 위해 사용되는 시스템의 프로세서의 개수와 토폴로지를 이용하여 태스크의 크기를 재구성하여 스케줄링 하는 것이다. 이를 위해, 주어진 태스크 그래프의 노드를 프로세서 수에 맞게 합병하거나 복제하여 제공되는 시스템 토폴로지를 표현한 시스템 그래프에서 가장 빠른 수행 시간을 얻을 수 있는 재구성 태스크 그래프(RTG)를 생성한다. 그리고 이를 직접적으로 프로세서에 스케줄링 함으로써 임의의 태스크 그래프에 대해 최적 혹은 준 최적의 결과를 얻을 수 있게 한다. 이 같이 할 경우 수행 시간이 절약되면서 간단한 스케줄링 처리가 된다.

기존의 연구와 비교하면 본 연구는 스케줄링을 하기 위해 단지 시스템 토폴로지에 대한 정보와 최초의 태스크 그래프를 프로세서 수와 토폴로지에 대한 정보를 이용하여 태스크 그래프 노드를 합병/복제하여 RTG를 구성하고, 이를 직접적으로 스케줄링을 한다. 이런 방법으로 스케줄링을 하면, 스케줄링을 한 후 다시 재 할당하거나 복제하는 작업이 필요 없게 되고, 완전 연결 토폴로지에서의 최적 결과에 대한 비교 분석 없이 최적 혹은 준 최적 결과를 산출할 수 있다.

2. 태스크 스케줄링

2.1 태스크 스케줄링

병렬 컴퓨터 사용시 성능 향상을 추구하는 기법 중 병렬 컴퓨터에 실행시키려는 프로그램(n개의 모듈로 구성)을 m개의 프로세서에 할당하는 것이 요구되는데, 이를 태스크 스케줄링이라고 한다[4, 5, 6]. 스케줄링 문제는 그 동안 이론적으로 많은 연구가 이루어져 왔고 이상적인 최적의 스케줄링을 하는 것은 NP-complete 문제로 알려져 있다[3]. 그러므로 최근에는 휴리스틱 알고리즘을 이용하여 빠른 시간 내에 스케줄링 할 수 있는 알고리즘에 대한 많은 연구가 이루어져 왔다[5, 9, 15].

휴리스틱 스케줄링에는 Hu가 제안한 알고리즘에

서, 선송시간을 고려한 경우와 고려하지 않은 경우이고, Yu가 제안한 알고리즘은 노드의 크기가 같다는 가정으로 스케줄링 알고리즘을 제안한 것이다. Kruatrachue에 의해 제안된 ISH(Insertion Scheduling Heuristic)은 준비된 태스크들을 유휴 시간(idle time)에 삽입함으로써 유휴 시간을 활용하고자 하는 알고리즘이고, DSH (Duplication Scheduling Heuristic)는 전송의 오프셋(offset)에 따라 태스크를 복사(duplicate)하는 알고리즘이다. 이들의 특징은 다음과 같다[3].

(1) 전송시간을 고려하지 않은 Hu의 알고리즘

이 알고리즘은 가정으로, 자료전송에 따른 지연을 고려하지 않은 스케줄링 알고리즘이다. 프로세서에 할당을 위한 태스크를 결정하기 그래프의 길이(length)를 고려하여 각 태스크의 우선 순위를 정한다. 태스크의 선택은 최상위 레벨 우선(highest-level-first)으로 우선 순위를 취한 뒤, 레벨에서 레벨(level-by-level)로의 방식을 행한다.

(2) 전송시간을 고려한 Hu의 알고리즘

전송시간을 고려하여 (1)의 알고리즘을 수정한 것이다. 태스크 선택 기준은 (1)의 알고리즘에서의 방법과 같다. 전송시간 지연은 임의의 프로세서가 선택되었을 때 선택된 태스크에 대한 시작 시간을 계산하는데 포함시킨다. 전송시간을 줄이기 위해 메시지를 보내는 태스크와 받는 태스크들을 각각 같은 프로세서에 할당한다. 새로운 태스크를 할당할 때 바로 앞의 태스크를 포함하고 있는 프로세서를 가장 먼저 고려하는 방법이다.

(3) ISH(Insertion Scheduling Heuristic)

이는 실행하고자 하는 태스크들을 프로세서의 유효 시간 슬롯(slot)에 삽입함으로써 사용하고 있지 않은 프로세서의 유효 시간을 활용하고자 하는 알고리즘이다. 이는 많은 요소들이 추가됨에 따라 현실적인 결과를 얻지만, 새로운 문제인 최대-최소 문제를 발생시킨다. 가장 긴 패스가 수행 시간뿐만 아니라 전송시간 지연에도 영향을 주기 때문에 프로세서를 더 추가함으로써 전체 수행 시간이 증가하는 결과가 나올 수가 있다.

(4) DSH(Duplication Scheduling Heuristic)

Kruatrachue에 의해 제안된 것으로, 최대-최소 문제를 태스크 복사 방법으로 해결한 것을 제외하고는 ISH와 유사하다. DSH는 1개의 앞선 태스크만을 복사하는 방법(Duplication of One Preceeding Task)과 앞의 모든 태스크를 복사하는 방법(Duplication of All Preceeding Task)으로 구분되는데, 전자는 각 태스크를 할당하기 전에 자료의 전송 지연에 영향을 주는 바로 앞의 태스크만을 복사하는 방법으로 수행 시간은 절약되지만, 스케줄링의 질을 떨어뜨리는 단점이 있다. 후자는, 완전 DSH 알고리즘이라고 하며, 자료의 전송 지연에 영향을 끼칠 수 있는 앞의 모든 태스크들을 복사시키는 방법이지만, 시간 복잡도가 $O(n^4)$이다.

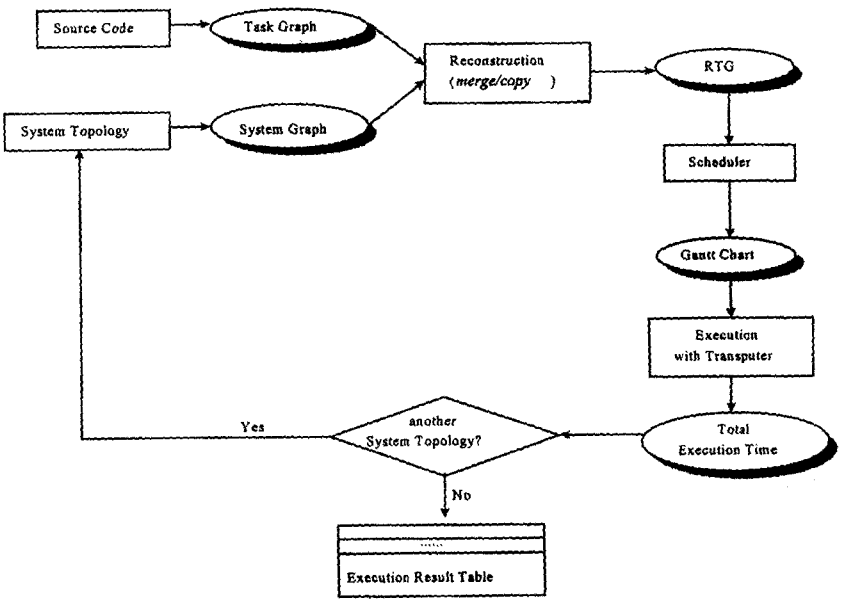
기존의 모든 스케줄링 알고리즘을 살펴보면, 태스크 스케줄링시 병렬성은 증가시키고 통신 오버헤드를 줄이면서 전체 수행 시간이 최소가 되도록 하는 것이 최적의 태스크 스케줄링이라고 밝히고 있다. 위의 (3), (4)의 스케줄링 방식은 일단 스케줄링이 이루어지고 난 후 그에 대한 정보를 이용하여 최적의 상태에 접근하도록 다시 스케줄링을 하는 방법을 취하고 있는데, 본 논문에서 제시하는 스케줄링 방식은 재할당 과정을 수행하지 않고 빠르고 간단한 스케줄링이 되도록 하는데 초점을 두고 있다.

3. 태스크 그래프의 재구성

본 논문은 최소 수행 시간을 얻기 위하여 태스크 그래프를 새롭게 구성한 재구성 태스크 그래프(RTG, Reconstructed Task Graph)로 정의하여 태스크 스케줄링을 한다.

3.1 재구성 태스크 그래프에 대한 태스크 스케줄링

본 논문에서 제시하는 태스크 스케줄링은 주어진 태스크 그래프의 각 레벨에 대해, 태스크 그래프의 노드 수를 프로세서 수와 일치되도록 노드를 합병하거나 복제하여 RTG를 구성하고, 이를 시스템 그래프에 직접 스케줄링 한다. 이 방식은 태스크 스케줄링 후에 다시 재 할당 작업 없이 빠른 스케줄링 시간, 간단한 스케줄링 방법, 그리고 준 최적의 수행 시간을 얻는다.



(그림 1) 재구성 태스크 그래프의 스케줄링 과정
(Fig. 1) Scheduling Step of Reconstructed Task Graphs

본 논문에서 제시하고자 하는 재구성 태스크 그래프의 스케줄링의 단계를 나타내면 (그림 1)과 같다.

본 연구에서 사용되는 태스크 그래프에 대한 특징은 태스크 그래프의 노드들 사이에 통신비용이 있는 경우를 고려하고, 태스크 그래프 노드들의 수행 시간에 상당한 차이가 없다고 보고, 또한 노드의 수행 시간과 노드와 노드 사이의 자료 교환에 필요한 통신비용도 큰 차이가 있지 않다고 가정을 한다.

본 논문에서 추구하는 스케줄링의 최종 목적은 주어진 태스크 그래프를 시스템 그래프의 정보를 이용하여 총 수행 시간을 최소화하도록 스케줄링을 하는 것이다. 즉, 주어진 태스크 그래프를 시스템 그래프에 정의된 홉 수의 값과 태스크 그래프의 노드간에 정의된 종속성과 그 종속성을 표현한 아크 위의 자료 전송 시간을 고려하여, 노드를 어떤 프로세서에 할당하여 처리하는 것이 가장 적은 수행 시간을 소비하고, 어떤 노드가 어떤 프로세서에 수행되는가에 따라 그 노드로 종속되어 있는 다른 노드들은 어떤 프로세서에 할당되는 것이 적절한 지 전체적인 값을 알 수 있어야 하며 이 전체적인 값이 태스크 그래프의 총 수행 시간이 된다.

또한 태스크 스케줄링은 병렬 처리 시스템 토폴로지가 존재하지 않는 상황에서 스케줄링이 이루어지는 경우 어떤 시스템 토폴로지에서 수행시키고자 하는 프로그램이 최적의 수행 시간을 제공하는 지에 대한 정보 또한 얻을 수 있어야 한다.

3.2 재구성 태스크 그래프(RTG)

각 레벨 노드의 수행 시간이 비슷하고, 각 노드로부터 자식 노드로의 통신비용이 비슷한 경우를 가정할 때, 태스크 그래프의 수행은 한 레벨의 노드 수행이 끝난 뒤, 그 레벨의 수행 결과가 다음 레벨에 전달되어야만 다음 레벨이 수행된다. 이 경우 통신비용도 비슷하므로 거의 같은 시간에 다음 레벨로 수행 결과가 전달된다. 이런 실행을 반복한 후에 태스크 그래프의 수행은 종료된다. 따라서, 본 논문에서는 프로세서 수와 각 레벨의 노드 수를 일치시킴에 따라 한 레벨을 프로세서에 할당하여 수행하고, 메시지 전송 후에 다음 레벨을 프로세서에 할당하는 방법으로 접근한다.

3.2.1 RTG

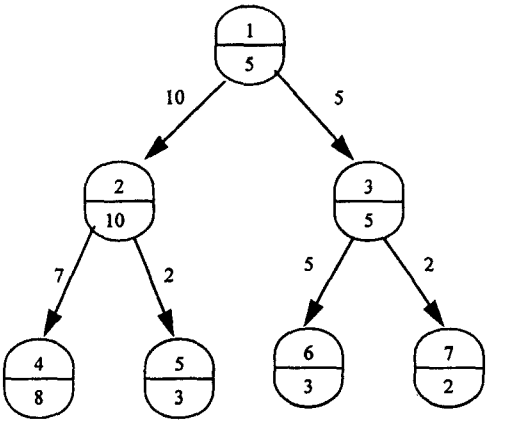
RTG는 태스크 그래프로부터 빠른 스케줄링 결과를 얻기 위해 재구성한 태스크 그래프이다. RTG의 특징은 각 레벨의 노드 수가 태스크 그래프를 수행시키고자 하는 프로세서의 수와 일치한다. RTG 각 레벨의 노드 수는 $|RTG_level_i|$로 정의한다. 이때, $|RTG_level_i| = |Vs|$가 성립된다.

시스템 그래프의 노드 수와 태스크 그래프 각 레벨의 노드 수를 일치하도록 구성한 태스크 그래프가 RTG이다. 이와 같이 태스크 그래프를 재구성하는 이유는, 가정으로 정의한 것과 같이 태스크 그래프의 각 레벨의 노드의 수행 시간이 큰 차이를 갖지 않고, 또한 각 노드의 자료 전송 시간이 거의 비슷한 경우에 태스크 그래프를 병렬 처리 시스템에 할당 수행되면, 각 레벨의 노드가 동시에 수행되고, 거의 같은 시간에 자료 전송이 이루어지기 때문이다. 또 다음 레벨의 노드가 수행이 되는 과정이 이루어 질 수 있기 때문에 태스크 그래프의 각 레벨의 노드를 시스템 그래프의 노드 수와 일치되도록 재구성했다.

RTG를 구성하는 기법은 합병/복제 방법에 의해 구성이 된다. RTG는 태스크 그래프와 유사하며 단지 합병/복제 방법에 의해 그래프가 재구성되기 때문에 노드 표현과 노드간의 종속성에 의한 통신비용의 표현 방법이 다소 차이가 있다. 합병 방법은 태스크 그래프의 임의의 레벨에서의 노드 수가 시스템 그래프의 노드 수보다 많이 존재하는 경우, 시스템 그래프의 노드 수와 일치하도록 노드들을 합병하는 과정이다. 복제 방법은 시스템 그래프의 노드 수보다 태스크 그래프의 임의의 레벨에서의 노드 수가 적은 경우 노드 수를 시스템 그래프의 노드 수와 같도록 복제하는 과정이다.

(그림 2)는 본 논문에서 추구하는 태스크 그래프 재구성에 대한 사항을 자세히 나타내 주고 있다.

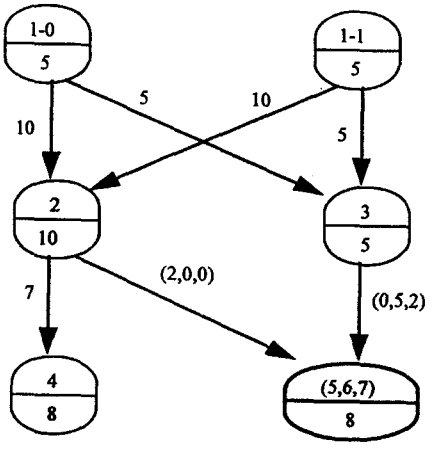
(그림 2(a))는 노드 7개로 구성된 태스크 그래프와 각 태스크 그래프 노드의 번호 및 각 노드의 수행 시간, 그리고, 노드간의 자료 전송 시간이 표현되어 있다. (그림 2(b))는 노드 2개로 구성된 시스템 그래프이다. sys_graph에서는 노드간의 자료 전송 지연 시간은 1로 규정되고, 자신의 노드에서의 자료 전송 시간은 무시함을 보여 주고 있다. (그림 2(c))는 (a)의 태스크 그래프를 (b)의 시스템 그래프에 스케줄링하기 위해 RTG로 구성한 결과를 나타내고 있다. RTG에서는



(a) 태스크 그래프

sys_graph ={ {0, 1},
{1, 0}}

(b) 시스템 그래프



(c) RTG

(그림 2) 태스크 그래프, 시스템 그래프와 RTG
(Fig. 2) Task Graph, System Graph and RTG

태스크 노드의 표현과 상위/하위 종속성에 대하여 통신 지연 시간의 표현이 태스크 그래프의 표현 방식과 다소 차이가 있다. 그러나, RTG로 표현되어 병렬 처리 시스템에서 수행된 프로그램의 결과와 재구성되기 전의 태스크 그래프를 수행시킨 결과는 차이가 없다. RTG로의 구성은 태스크 그래프를 병렬 처리 시스템에서 효율적으로 수행시키기 위한 방법으로 제공된 것이다. 즉, RTG는 병렬 처리 시스템에서 태스크 그래프의 노드가 최적의 상태에서 가장 적은 수행 시간을 가질 수 있도록 재구성된 태스크 그래프이다.

태스크 그래프로부터 RTG를 구성하기 위한 합병과 복제를 위한 기본 알고리즘은 먼저 태스크 그래프의 각 레벨에 대하여 프로세서의 수와 각 레벨의 노드 수를 비교하여, 노드의 수가 프로세서의 수보다 많으면($|task\_level_i| > |Vs|$) 그 레벨은 프로세서의 수와 같은 노드 수가 되도록 합병한다. 또, 노드의 수가 프로세서의 수보다 적은 경우($|task\_level_i| > |Vs|$)에 있어서는 해당 레벨을 복제를 이용하여, 각 레벨의 노드의 수를 프로세서의 수와 같도록 한다. 합병과 복제를 이용한 RTG의 구성은 태스크 그래프의 루트 레벨로부터 태스크 그래프의 하위 노드로 수행한다.

RTG의 기본 알고리즘은 먼저 입력으로 태스크 그래프를 받아들여 이 태스크 그래프로부터 레벨을 계산하고, 각 레벨이 몇 개의 노드로 구성되어 있는가, 어떤 노드가 어느 레벨에 속하는가에 대한 정보를 얻어내는 정보 테이블을 구성한다. 레벨의 수가 정의되므로 태스크 그래프의 각 레벨에 대하여 시스템 그래프의 노드 수와 레벨의 노드 수를 비교하여 합병 과정과 복제 과정을 수행한다. 이 과정은 하위 레벨까지 수행한다. 수행 중 잎 노드(leaf node)는 이 과정에 포함시키지 않는다.

3.2.2 노드의 합병

1) 합병 대상 노드의 선택

합병 알고리즘은 태스크 그래프의 각 레벨에 대해서 노드의 수가 프로세서의 수보다 많은 경우에 수행한다. $|task_level_i| > |Vs|$인 경우 노드를 합병하기 위한 노드 선택 기준은 상위 종속성을 갖는 노드를 대상으로 다음과 같이 적용한다.

① 레벨의 노드 수행 시간 중 가장 적은 두 노드를 선택한다.

$$m\_node_i = SELECT(min_i\{e(n_i)\})$$

② 레벨의 노드 수행 시간과 자식 노드로의 통신비용 합의 결과가 가장 적은 두 노드를 선택한다.

$$m\_node_i = SELECT(min_i\{e(n_i) + c(n_i, n_j)\})$$

합병을 필요로 하는 레벨에서 합병하기 위한 노드 선택시 노드 수행 시간이 가장 적은 두 노드를 선택하는 이유는 노드 수행 시간이 가장 작은 두 노드를 합병하여 하나의 노드로 구성하면 해당 레벨에 속해 있는 노드들의 수행 시간과 일치시킬 수 있기 때문이다. 또한 여러 개의 노드가 합병 대상 노드로 선택이 되는 경우에 있어서는 해당 노드들의 수행 시간과 해당 노드에서 자식 노드로의 자료 전송시 요구되는 통신비용의 합을 구하여 그 중 가장 적은 값을 갖는 두 노드를 선택한다. 이 때에도 여러 개의 노드가 대상이 되면, 자식 노드를 많이 갖고 있는 노드, 즉 하위 종속성을 많이 지닌 노드를 선택한다. 이 경우에도 여러 대상 노드가 있으면 본 논문에서는 노드 번호가 작은 것을 대상으로 선택한다. (그림 3)

```
MERGE_node()
  {
    sort : ascending : the execution time of each node
    select : (1) m_node_i = SELECT( min_i{ e(n_i) } )
             (2) m_node_i = SELECT( min_i{ e(n_i) + c(n_i, n_j) } )
    merge :- sum of execution time
                    : TET = Σ_{i=0} e(n_i)
            :- node : (n_i, n_j, n_k, ... )
            :- maintain the communication cost
                    if(n_i, n_u : upper dependency) then c(n_u, n_i)
                    if(n_i, n_l : lower dependency) then c(n_i, n_l)
    if ( |Vs| == |task_level_i| )
        quit;
    else
        MERGE_node();
    endif
  }
```

(그림 3) 합병 알고리즘
(Fig. 3) Algorithm of Node Merge

2) 합병된 노드의 특성

합병된 노드는 태스크 그래프의 노드 특성을 그대로 지닌다. 즉, 합병된 노드는 노드 번호, 수행 시간, 상위 종속성, 그리고 하위 종속성을 갖는다.

합병된 노드의 표현은 노드 번호를 괄호 안에 (i, j, k) 형태로 표현한다. 이는 같은 레벨의 노드 i, j, k가 하나의 노드로 합병됨을 의미한다. 합병된 노드의 수행 시간은 태스크 그래프의 수행 시간을 그대로 유지하고, 또한 각 수행 시간의 합을 합병 노드의 수행 시간으로 사용한다. 합병된 노드의 수행 시간은 스케줄링 시 합병된 노드가 수행되어야 할 수행 시간이 되며, 하나의 프로세서에서 해당 수행 시간만큼 수행된다. 그리고, 태스크 그래프의 수행 시간을 그대로 유지하는 이유는 합병된 노드가 프로세서에 할당시, 합병된 노드에 속한 태스크 그래프의 노드는 수행 순서를 정의하여 수행할 수 있다. 상위 종속성에 의해 합병된 노드들 중에서 수행에 필요한 자료의 값이 전송되면, 값을 받는 노드는 먼저 수행이 된다.

합병된 노드에서 통신비용에 대한 표현은 상위 종속성에 대한 통신비용 표현과 하위 종속성에 대한 통신비용 표현이 있다. 두 경우 모두 같은 형태로 표현된다.

3.2.3 노드의 복제

노드 복제는 복제 대상 레벨의 노드 중 임의의 노드를 복제하여 레벨에 추가하는 것이다. 복제된 노드는 원래 노드의 역할을 그대로 유지한다. 단 노드의 번호 표현은 원래 노드와 복제된 노드 모두 바꾼다. 그러나 원래 노드가 갖고 있던 노드의 수행 시간, 상위 종속성, 및 하위 종속성은 그대로 유지되며 표현 형태도 동일하다.

1) 복제 대상 노드 선택

각 레벨에 대하여 노드 수가 프로세서 수보다 적은 경우에 프로세서 수만큼 해당 레벨의 노드 수를 복제 대상 레벨로 선정하여 프로세서 수와 같은 노드 수가 되도록 복제한다. 즉, $|task_level_i| < |Vs|$인 경우 다음의 사항을 고려하여 복제한다.

① 잎 노드는 복제하지 않는다.

② 복제는 $|Vs| - |task\_level_i|$ 만큼 한다.

③ 복제된 노드의 수행 시간, 노드간의 종속성, 통신비용은 유지한다.

노드를 복제하기 위한 대상 노드 선택의 경우 노드의 수행 시간과 노드의 하위 종속성을 고려하여 복제한다. (그림 4) 가정에서 RTG의 기본 골격은 각 레벨 노드의 수행 시간이 거의 비슷한 수행 시간을 갖는 태스크 그래프로 정의하였으나 복제 시에는 가장 큰 통신비용을 갖는 노드, 또는 하위 종속성을 많이 갖는 노드를 대상으로 복제한다. 큰 통신비용을 갖는 노드를 선택하는 것은 전체 태스크 그래프의 수행 시간에 있어서 수행 시간이 큰 노드를 여러 프로세서에서 수행시킬 경우 전체 수행 시간을 절약할 수 있기 때문이다. 또 하위 종속성이 많은 노드를 복제 대상 노드로 선택하는 이유는 복제 대상 노드가 여러 프로세서에서 수행되어 많은 하위 노드로 자료 전송을 빠르게 하기 위함이다.

```
COPY_node()
    {
      if ( node != leave node )
       {
         sort : descending : the execution time of each node
         select : c_node_i = SELECT( max_i( e(n_i) ) )
         if( select two more)
              select the largest child node
         copy : |Vs| - |task_level_i|
              - node : n_i-0, n_i-1, n_i-2, .... , n_i-j-1
              - execution time : e(n_i)=e(n_i-0)=e(n_i-1)= ....
                =e(n_i-j-1)
              - communication cost : c(n_i, n_k)=c(n_i-0, n_k)
                =c(n_i-1, n_k)= .... =c(n_i-j-1, n_k)
              - arc
       }
    }
```

(그림 4) 복제 알고리즘
(Fig. 4) Algorithm of Node Copy

2) 복제된 노드의 특성

복제된 노드는 태스크 그래프 노드의 특성을 그대로 유지하지만, 노드 번호, 노드 수행 시간, 상위 종속성, 하위 종속성, 그리고 자료 전송 시간의 표현에 다소 차이가 있다. 복제된 노드는 다음과 같이 표현한다.

① 복제 노드는 다음과 같이 표기한다.

노드 n_i가 j개의 노드로 복제되는 경우 표현은 다음과 같이 한다.

$$n_{i-0}, n_{i-1}, n_{i-2}, \ldots, n_{i-j-1}$$

② 복제 노드의 상위 종속성, 하위 종속성, 그리고 통신비용은 유지하고, 표현 방법은 태스크 그래프와 동일하다.

3) 노드 복제의 과정

노드의 복제 과정은 앞의 정의에서와 같이 노드 번호, 노드 수행 시간, 상위 종속성, 하위 종속성, 그리고 자료 전송 시간을 고려하여 복제한다.

① 노드가 하나인 경우

태스크 그래프의 임의의 레벨에 노드가 하나밖에 없는 경우에는 해당 노드를 시스템 그래프 노드 수만큼 복제한다. 즉, |Vs|-1만큼 노드를 복제한다.

② 잎 노드의 경우

시스템 그래프의 노드 수보다 적지만 복제하지 않는다. 만약 복제를 하게 되면, 복제된 노드를 수행시키게 되어 수행 시간의 지연 결과를 초래하게 된다.

③ 여러 개의 노드 중에서 복제 대상 노드를 선택하는 방법

임의의 레벨에 시스템 그래프의 노드 수보다 적은 노드가 존재하여 복제하게 될 경우 하위 종속성을 많이 지닌 노드를 대상 노드로 선택하여 복제한다. 이 노드들이 여러개 존재하는 경우에는 노드들을 수행 시간에 의하여 내림차순으로 정렬한 후 이 중에서 가장 큰 수행 시간을 갖는 노드를 복제 대상 노드로 선택하여 복제를 수행한다.

4. RTG의 수행 및 결과 분석

본 연구에서 제시하는 개선된 스케줄링을 위해서는 먼저, 사용자 프로그램으로부터 태스크 그래프를 구성하고, 이를 합병/복제 방법에 의하여 재구성된 태스크 그래프 RTG를 구성한다. 그리고 시스템 토폴로지로부터 시스템 그래프를 구성한다. RTG를 구성할 때에는 시스템 그래프의 정보를 이용한다. 구성된 RTG와 시스템 그래프를 이용하여 스케줄링을 수행해 할당된 결과를 간트 차트로 얻는다.

또 스케줄링이 수행된 후에는, 스케줄링된 결과를 직접 병렬 처리 시스템인 트랜스퓨터에서 수행시켜 수행 결과를 얻는다. 트랜스퓨터에서 수행시킨 결과가 스케줄링된 결과와 일치하는 지 확인한다. 그리고 트랜스퓨터에서 구성이 가능한 시스템 토폴로지에 대하여 RTG를 재구성 및 스케줄링을 하고, 트랜스퓨터에서 수행시킨다.

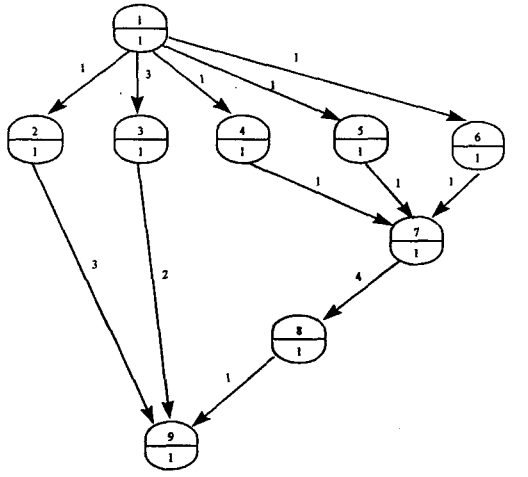
RTG 스케줄링과 다른 알고리즘과의 비교 분석을 위해 기존 논문[3]에서 제시된 스케줄링 알고리즘의 결과와 본 논문에서 제시한 RTG 스케줄링을 이용하여 RTG를 구성하고, 스케줄링한 결과를 나타냈다.

4.1 기존의 태스크 스케줄링과의 비교

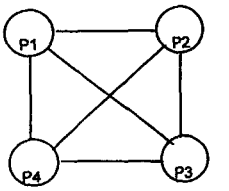
(1) 태스크 그래프와 시스템 그래프

비교 분석하기 위한 태스크 그래프는 Kruatrachue 논문[3]에서 제시된 태스크 그래프로서 (그림 5)에 나타나 있다.

태스크 그래프의 노드 수는 9개이고, 각 태스크 노드의 수행 시간은 동일하다(수행 시간=1). 노드와 노드 사이에는 통신 지연 시간이 존재한다. 시스템 그래프는 4개의 프로세서로 구성되있고, 시스템 토폴로지는 완전 연결 구조로 되어 있다. 즉, 각 프로세서간의 전송 시간은 1이다.



(a) 태스크 그래프



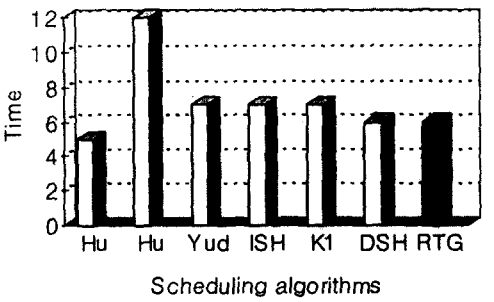
sys_graph ={{0, 1, 1, 1},
{1, 0, 1, 1},
{1, 1, 0, 1},
{1, 1, 1, 0}}

(b) 시스템 그래프

(그림 5) 태스크 그래프와 시스템 그래프
(Fig. 5) Presentation of Task Graph and System Graph

(2) 기존 스케줄링 알고리즘들의 결과

(그림 5)의 태스크 그래프에 대하여 여러 태스크 스케줄링 알고리즘을 적용하였다. 비교 분석하기 위하여 사용한 태스크 스케줄링 알고리즘은 Hu의 태스크 스케줄링 알고리즘, Hu의 통신 지연 시간을 고려한 태스크 스케줄링 알고리즘, Yud의 태스크 스케줄링 알고리즘, ISH 알고리즘, K1 알고리즘, 그리고 DSH 알고리즘 등이다. 스케줄링의 결과는 간트 차트로 나타냈다. (그림 6)는 각 태스크 스케줄링에 대한 결과를 표시한 것이다.



(그림 6) 태스크 스케줄링 알고리즘의 비교 분석 그래프
(Fig. 6) Results of Performance

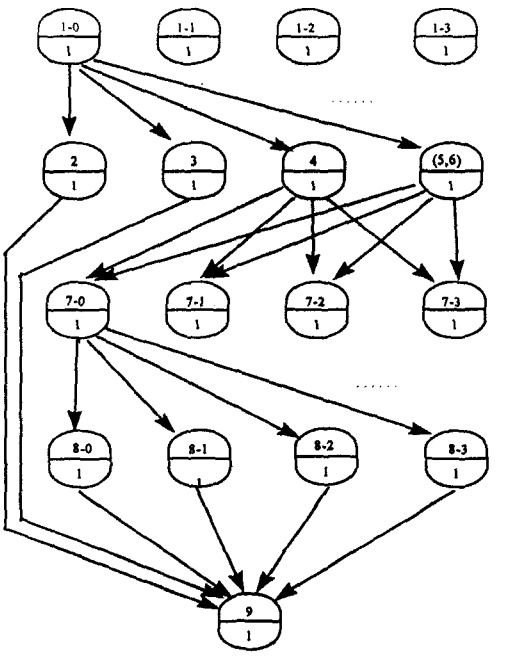
(3) RTG와 스케줄링 결과

주어진 태스크 그래프를 합병/복제 방법을 이용하여, RTG를 구성한 결과가 (그림 7(a))에 나타나 있다. 시스템 그래프의 노드가 4개이므로 이에 맞게 태스크 그래프의 각 레벨 노드 수를 4개로 구성한 것이다. 복제된 노드의 표현은 1-0, 1-1, 1-2, 1-3으로 표현하였고, 합병된 노드는 (5, 6)으로 표현하였다.

RTG로 구성된 그래프를 RTG 스케줄링 알고리즘을 이용하여, 스케줄링한 결과를 (그림 7(b))의 간트 차트로 나타냈다.

시스템 그래프의 노드 수에 맞춰 복제된 노드들이 각 프로세서에 할당되고, 이들의 수행 결과에 따라 다음의 노드들이 적절히 스케줄링 알고리즘에 의하여 프로세서에 배치된다. 구성된 RTG 그래프는 RTG 스케줄링에 의해 전체 수행 시간이 단위 시간 6으로 수행됨을 결과로 보여 준다.

Hu의 알고리즘은 태스크 그래프의 노드들 사이에



(a) RTG

Processors

Time	1	2	3	4
0	1-0	1-1	1-2	1-3
1	2	3	4	5
2				6
3				7-0
4	7-1	7-2	7-3	8-0
5	8-1	8-2	8-3	9
6				

(b) RTG 스케줄링 결과

(그림 7) RTG와 스케줄링 결과
(Fig. 7) RTG and Results of RTG Scheduling

전송 지연이 존재하지 않는 경우 전체 수행 시간은 단위 시간 5가 걸린다. 그러나, (그림 5)에서 보여지는 태스크 그래프는 태스크 그래프 노드들 사이에 통신 지연 시간이 존재하므로, Hu가 제시한 통신 지연 시간이 존재하지 않는 알고리즘은 의미가 없게 된다. 그리고 Hu의 또 다른 전송 지연 시간을 고려한 경우

단위 시간 12만큼의 시간이 소요된다.

기존에 제시된 알고리즘 중 DSH 알고리즘이 단위 시간 6으로 가장 작은 전체 수행 시간을 나타내고 있다. 또한 DSH 알고리즘은 태스크 그래프의 전체 수행 시간을 줄이기 위해 시스템 그래프의 프로세서들을 최대한 활용하여, 스케줄링 하였음을 보여 주고 있다.

본 논문에서 제시한 RTG 스케줄링 알고리즘은 DSH와 같은 단위 시간 6으로 전체 수행 시간을 나타내 주고 있다. 그리고 전체 수행 시간을 줄이기 위하여 시스템 그래프의 노드들을 최대한 활용하였다.

기존의 태스크 스케줄링 알고리즘과 비교하여, RTG 스케줄링 알고리즘은 전체 수행 시간 중 가장 적은 수행 시간을 나타내 주고 있고, 같은 전체 수행 시간을 나타낸 DSH 알고리즘과 비교시 빠른 스케줄링 결과를 제시하는 장점을 나타낸다. DSH 알고리즘은 태스크 스케줄링이 이루어진 후 다시 스케줄링을 취하는 방법으로 전체 수행 시간을 얻었으므로 빠른 스케줄링 방법은 아니다. 그러나 RTG 스케줄링은 RTG 태스크 그래프가 구성될 경우 그 결과를 직접 스케줄링 하므로 전체 수행 시간을 빨리 구할 수 있다.

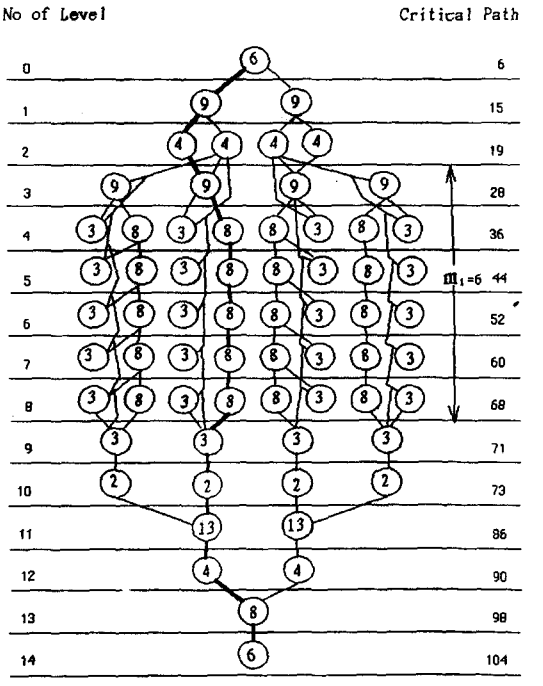
4.2 RTG를 이용한 암호화 알고리즘의 수행 및 결과 분석

(1) 태스크 그래프와 시스템 그래프

병렬 처리 시스템 환경에서 RSA 암호화를 고속 병렬 처리하기 위하여 프로그램을 분석하여 수행하였다. 특히, 암호화(복호화)의 속도를 향상시키기 위해서 키의 크기가 큰 복호화의 경우 비밀로 유지되는 N=P*Q를 P와 Q로 분해하여 암호문 C에 대해 병렬 중국인 나머지 정리 알고리즘을 적용할 수 있다[17]. M=CD mod N (N=P*Q)을 계산하기 위해서 중국인 나머지 정리 알고리즘을 이용한 병렬 구현 알고리즘은 다음 (그림 8)과 같다. 앞의 (그림 8)을 복호화 예를 바탕으로 실제 수행되는 기계 명령어를 추적하여, 기본 블록으로 구성한 후 기본 블록들 간의 병렬성을 분석, 추출하여 자료흐름 및 제어흐름 그래프로 나타낸 것이 (그림 9)의 태스크 그래프이다. 이 그래프는 기본 블록을 통한 제어 흐름의 가능한 경로도 포함하고 있다.

```
parbegin;
        Ap = C (mod P);
        Aq = C (mod Q);
parend;
parbegin;
        Bp = A^Dp (mod P);
        Bq = A^Dq (mod Q);
parend;
parbegin;
        Mp = Bp * Cp (mod N);
        Mq = Bq * Cq (mod N);
parend;
M = Mp + Mq;
if( M >= N ) M = M - N
else M = M;
return(M);
```

(그림 8) 병렬 중국인 나머지 정리 알고리즘
(Fig. 8) Test Algorithm : Parallel Chinese Remainder Theory



(그림 9) 병렬 중국인 나머지 정리 알고리즘의 태스크 그래프
(Fig. 9) Task Graph : Parallel Chinese Remainder Theory

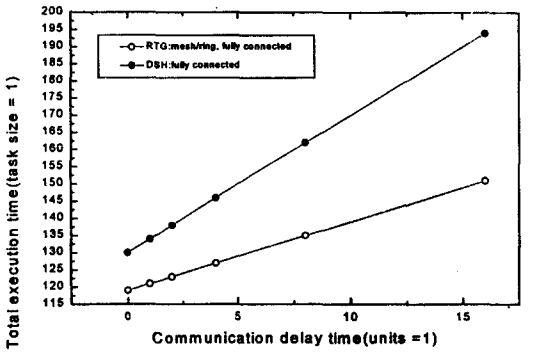
(2) RTG와 스케줄링 결과

병렬 중국인 나머지 정리 알고리즘에 대하여 RTG의 구성은 4개의 프로세서와 8개의 프로세서에 대하여 구성하였다. 병렬 중국인 나머지 정리 알고리즘은

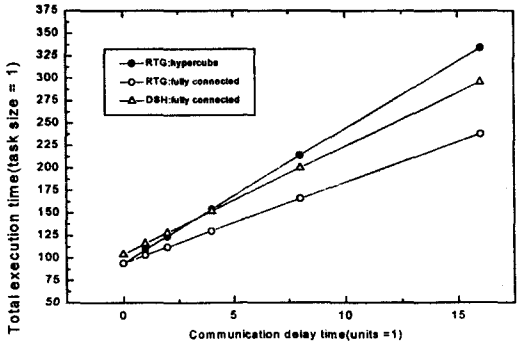
상위 레벨과 하위 레벨에 있어서 태스크 그래프 노드의 구성이 거의 유사하여 간단히 RTG의 구성이 되었고 4개의 프로세서 구성시 레벨 4에서 레벨 8까지 노드들만 합병되는 과정을 보여 주었다.

4개로 제공되는 시스템 그래프의 노드에서 메쉬 구조의 상태와 완전 연결 상태에 대하여 스케줄링을 하였다. 8개의 시스템 그래프의 노드에 대해서는 하이퍼큐브 상태와 완전 연결 상태에 대하여 스케줄링 테스트를 하였다.

RTG 스케줄링의 결과는 4개의 프로세서 노드를 갖는 시스템 그래프와 8개의 노드를 갖는 시스템 그래프에 대하여 나타내었다. 이 때, 적용한 변이는 전송 지연 시간을 사용하였다. 태스크 그래프의 노드와 노드 사이의 전송 시간이 0, 1, 2, 4, 8, 16인 경우에 대하여 스케줄링을 하였다. 스케줄링을 수행한 결과가



(a) 4개의 프로세서에서 수행된 결과



(b) 8개의 프로세서에서 수행된 결과

(그림 10) 암호화 알고리즘의 스케줄링 결과

(Fig. 10) Results of Performance : Parallel Chinese Remainder Theory

(그림 10)에 나타나 있다. (그림 10 (a))에서와 같이 병렬 중국인 나머지 정리 알고리즘을 수행시키기 위한 최적의 병렬 환경은 4개의 프로세서로 구성된 완전 연결 구조이다. 테스트 한 알고리즘의 특성상 프로세서 수가 증가하는 경우에는 단위 시간이 증가하는 결과를 가져 왔다.

RTG로 스케줄링한 결과가 DSH로 스케줄링한 결과에 비해 성능이 향상되어 있음을 알 수 있다. 이는 DSH가 스케줄링을 하고난 후 재 할당하는 과정에서 할당된 태스크들 사이에 재 할당하다가 이루어지기 때문에 성능이 다소 떨어진 결과를 나타냈다.

5. 결 론

본 논문에서 제시한 스케줄링 기법은 병렬 수행시키려는 많은 프로그램 중 제어/자료 종속성에 의해 병목 현상이 존재하는 경우에 있어 효과적으로 병렬성을 얻을 수 있는 방법을 제시하였다. 병목 현상이 존재하는 부분 때문에 전체 프로그램 수행에 영향을 미치는 경우, 이를 병렬 수행시키고자 하는 시스템 토폴로지의 정보를 이용하여 RTG로 변환하여 태스크 그래프를 재구성 한 후 스케줄링 하면 기존에 제시된 스케줄링 알고리즘보다 좋은 스케줄링 결과를 얻었다. RTG의 특성이 스케줄링 하고자하는 병렬 처리 시스템의 시스템 토폴로지의 형태를 최대한 고려하여 재구성 된 프로그램 모듈이기 때문에 병렬 수행의 효과를 최대로 얻을 수 있었다.

앞으로 태스크 그래프를 재구성하여 RTG로 만들 때 태스크 그래프의 레벨만을 고려하여 재구성을 하였으나, 태스크 그래프의 상위/하위 종속성을 고려하여 여러 레벨을 이용하여 RTG를 구성하면 더 효과적으로 병렬 수행이 될 수 있는 RTG가 만들어 질 것으로 사료된다.

참 고 문 헌

[1] T. L. Adam, K. M. Chandy and J. R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," Communications of the ACM, Vol. 17, No. 12, pp. 685-690, December 1974.

[2] R. G. Cytron, "Compile-Time Scheduling and Optimization for Multiprocessor Systems," Ph. D. dissertation, Dept. of Computer Science, University of Illinois, September 1984.

[3] B. Kruatrachue, "Static Task Scheduling and Grain Packing in Parallel Processing Systems," Ph. D. dissertation, Oregon State University, Corvallis, Oregon, 1987.

[4] S. H. Bokhari, "On the Mapping Problem," IEEE Transactions on Computers, Vol. C-30, No. 3, pp. 207-214, March 1981.

[5] H. El-Rewini, and T. G. Lewis "Scheduling Program Tasks onto Arbitrary Target Architecture," Journal of Parallel and Distributed Computing, Vol. 9, pp. 138-153, 1990.

[6] R. Agrawal, H. V. Jagadish, "Partitioning Techniques for Large-Grained Parallelism," IEEE Transactions on Computers, Vol. 37, No. 12, pp. 1627-1634, December 1988.

[7] F. D. Anger, J. J. Hwang, and Y. C. Chow, "Scheduling with Sufficient Loosely Coupled Processors," Journal of Parallel and Distributed Computing, Vol. 8, pp. 87-92, 1990.

[8] J. Yang, L. Bic and A. Nicolau, "A Mapping Strategy For MIMD Computers," 1991 International Conference on Parallel Processing, Vol. I, pp. 102-109, 1991.

[9] T. G. Lewis, and H. El-Rewini, Introduction to Parallel Computing, Prentice-Hall International Editions, 1992.

[10] T. Baba, Y. Iwamoto and T. Yosshinaga, "A Network-Topology Independent Task Allocation Strategy for Parallel Computers," Proceedings Supercomputing '90, pp. 878-887, 1990.

[11] K. P. Belkhale, "Approximate Algorithms for the Partitionable Independent Task Scheduling Problem," 1990 International Conference on Parallel Processing, Vol. 1, pp. 72-75, 1990.

[12] C. McCreary and H. Gill, "Efficient Exploitation of Concurrency using Graph Decomposition," 1990 International Conference on Parallel Processing, Vol. II, pp. 199-203, 1990.

[13] C. D. Polychronopoulos, Parallel Programming and Compilers, Kluwer Academic Publishers.

[14] S. S. Yau and V. R. Satish, "A Task Allocation Algorithm for Distributed Computing Systems," COMPSAC, 1993.

[15] I. Ahmad and Y. K. Kwok, "A New Approach to Scheduling Parallel Programs Using Task Duplication," 1994 International Conference on Parallel Processing, Vol. II, pp. 47-51, August 1994.

[16] A. Gerasoulis and T. Yang, "On the Granularity and Clustering of Directed Acyclic Task Graph," IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 6, pp. 686-701, June 1993.

[17] M. Shand and J. Vuillemin, "Fast Implementations for RSA Cryptology," 12-th IEEE Symposium on Computer Arithmetic, 1993.

변 승 환

1988년 충남대학교 계산통계학과 졸업(이학사).
1990년 충남대학교 대학원 계산통계학과 졸업(이학석사).
1996년 충남대학교 대학원 전산학과 졸업(이학박사).
1994~현재 서남대학교 전산정보학과 전임강사.
관심분야:병렬처리, 멀티미디어 응용, 에이전트 시스템.

유 관 종

1976년 서울대학교 계산통계학과 졸업(이학사).
1978년 서울대학교 대학원 계산통계학과 졸업(이학석사).
1984년 서울대학교 대학원 계산통계학과 졸업(이학박사).
1989년~1990년 캘리포니아 대학(IRVINE) 방문 교수.
1979~현재 충남대학교 컴퓨터과학과 교수.
관심분야:프로그래밍 언어, 병렬처리, 멀티미디어 응용.