통신의 영향을 줄이기 위한 이기종 태스크 스케쥴링 기법

문 현 주† · 전 중 남†† · 김 석 일†† · 황 인 재†††

요 약

본 논문에서는 이기종 머신으로 구성된 분산환경에서 분산처리를 하는 경우에 발생하는 태스크간의 과도한 통신오버헤드를 완화할 수 있는 이기종 복사 스케쥴링(HDS:Heterogeneous Duplication Scheduling) 기법을 제안하였다. HDS 기법은 서로 다른 머신에 할당된 두 태스크간에 과도한 통신이 발생하는 경우, 통신을 유발하는 태스크를 데이터를 전송받을 태스크가 할당될 머신에 복사하여 함께 수행하는 기법이다. 이 기법에서는 복사할 태스크의 범위를 부모 태스크로 한정하여 알고리즘의 복잡도를 낮추었다. 여러 형태의 태스크 그래프에 대한 모의실험에서도 제안한 기법에 의한 스케쥴링 결과가 기존의 이기종 스케쥴링 기법에 의한 결과에 비하여 우수한 것을 확인할 수 있었다.

A Heterogeneous Task Scheduling Reducing Effects of Communication

Hyun-Ju Moon† · Joong-Nam Jeon†† · Suk-Il Kim†† · In-Jae Hwang†††

ABSTRACT

This paper proposes Heterogeneous Duplication Scheduling(HDS) which alleviates excessive communication overhead between tasks for distributed computing on a heterogeneous distributed environment. HDS is to allocate a copy of a task that causes excessive data communication with a message receiving task to the same machine wherein the message receiving task is scheduled. The proposed algorithm allows only the duplication of parent tasks so as not to increase the complexity of the algorithm. Simulation on various type of task graphs provides that the scheduling results by using HDS are better than those by using the existing heterogeneous scheduling schemes.

1. 서 론

분산처리는 널리 산재해 있는 연산자원들을 빠른 네트워크로 연결하여 응용 프로그램을 처리하는 기법이다. 따라서 분산처리는 하나의 응용 프로그램을 몇 개의 태스크로 분할하고 이들 태스크들을 서로 다른 머신에 할당하여 동시에 수행하도록 하므로써 응용 프로그램을 빠르게 처리할 수 있다. 그러므로 분산처리를 함에 따른 성능 이득은 응용 프로그램을 많은 태스크로 분할하고 이들 태스크를 보다 많은 머신을 이용하여 얼마나 균형있게 배치하는가와 머신간의 통신 비용을 얼마나 줄일 수 있는가에 달려있다. 이러한 문제는 결국 태스크를 머신에 할당하는 기법인 스케쥴링 문제로 귀착된다.

분산처리에 있어서 스케쥴링에 관한 연구는 머신 및 응용 프로그램에 대한 정보를 이용하여 컴파일시에

※ 이 논문은 1997년 한국학술진흥재단의 공모과제 연구비에 의하여 연구되었음
† 준 회 원 : 충북대학교 대학원 컴퓨터과학과
†† 종신회원 : 충북대학교 컴퓨터과학과 교수
††† 종신회원 : 충북대학교 컴퓨터교육과 교수
논문접수 : 1997년 12월 31일, 심사완료 : 1998년 7월 15일

태스크의 머신 할당을 결정하는 정적 스케쥴링(static scheduling) 방법과 실시간에 변화하는 분산환경의 상태를 반영하여 실행시간(runtime)에 주기적으로 대처하도록 하는 동적 스케쥴링(dynamic scheduling) 방법으로 구분된다. 동적 스케쥴링에 관한 연구는 머신들의 사용가능여부가 불규칙적으로 변화하는 환경에서 머신간의 부하 균형을 유지하는 기법에 관한 연구[1,2]와 부하 균형을 유지하기 위하여 부하가 큰 머신으로부터 부하가 작은 머신으로 태스크를 이동하는 방법에 관한 연구[3,4]로 나뉘어 진행되어 왔다. 결국 동적 스케쥴링 기법은 머신간의 태스크 이동이 가능한 환경이 갖추어지셔야 하므로 태스크의 이동으로 인한 코드의 변환이 불필요한 동일 기종 환경[5,6]이나 태스크의 실행에 필요한 데이터들을 머신 상호간에 이동시킬 수 있는 데이터 병렬성에 관한 연구[7,8]로 국한되어 왔다. 근래에는 Java등과 같이 머신의 종류에 관계없이 같은 코드를 사용하는 환경에서 이기종 분산처리를 하기 위한 연구도 진행 중이다[9,10].

이와는 달리 정적 스케쥴링 기법은 컴파일시에 태스크별로 할당될 머신이 결정되므로 각각의 태스크가 머신별로 사전에 구획되어 각각의 머신에서 컴파일된다. 따라서 부하의 이동에 따른 코드의 변환이 불필요한 장점이 있으나 실행시 여러 가지 이유로 발생하는 심각한 부하의 불균형을 대처하는 수단이 없으므로 컴파일시에 예측한 내용들이 실행시에도 유지되어야 하는 단점이 있다. 또한 정적 스케쥴링 기법에 대한 연구는 어떤 응용 프로그램을 분산처리하는 동안에는 시스템을 구성하고 있는 머신들이 이 작업을 위해서 전적으로 사용되는 것으로 간주한다. 따라서 실제 환경에 적용할 경우, 약간의 부하 불균형에 의해서도 전체 시스템의 성능이 저하되는 단점이 있다. 이러한 단점에도 불구하고 동적 스케쥴링 기법이 지니는 여러 가지 제약들로 인하여 정적 스케쥴링에 관한 연구가 활발히 진행되고 있다.

분산환경을 구성하는 머신들이 동일한 성능과 기능을 지닌 동일 기종 환경에서 정적 스케쥴링 기법을 사용하여 최적의 결과를 얻을 수 있는 스케쥴링 문제는 NP-hard 문제의 하나이다[11]. 더구나 시스템을 구성하는 머신들이 서로 상이한 성능과 특성을 지닌 이기종 머신들인 경우에는 동일 기종 환경보다 고려할 사항이 많아진다. 예를 들면, 머신간에 자료 전송이 일어나는 경우에 각각의 머신내에서 사용하는 자료의 저장 형식이 다르다면 목적지 머신의 자료 저장형식에 맞도록 자료를 변환하여 전송하거나 수신 후 이를 변환하여야 한다. 그러나 이 문제는 PVM(Parallel Virtual Machine)[24]이나 MPI(Message Passing Interface)[25]와 같이 자동적인 저장 형식 변환을 제공하는 가상 환경을 사용하므로써 해결이 가능하다. 또한, 이기종 분산환경에서는 각 태스크들이 그 특성에 따라 어느 특정한 머신에서는 매우 빠르게 수행되는가하면 어떤 머신에서는 실행이 늦어지게 되는 경우가 발생하기도 한다. 이러한 여러 가지 요소들로 인하여 이기종 환경에서의 정적 스케쥴링 문제도 NP-hard 문제이며, 최적에 근접한 결과를 얻기 위한 경험적 스케쥴링 기법들이 다양하게 연구된 바 있다[12-17].

본 논문에서는 기존의 병렬머신 환경에서 연구된 태스크 복사 기법[18]을 이기종 환경의 특성 및 알고리즘의 가용성을 고려하여 확장한 정적 스케쥴링 기법을 연구하였다. 태스크 복사 기법은 어떤 태스크를 할당하는 시점에서 이 태스크의 수행을 앞당겨 시작할 수 있는 경우에 한하여 태스크 그래프의 부모노드를 복사하는 기법이다. 이와 같은 부모노드의 복사는 병렬환경에 비하여 통신오버헤드가 큰 분산환경에 효과적으로 적용될 수 있다. 본 논문에서 제안한 스케쥴링 기법은 기존의 경험적 스케쥴링 기법들과 같이 두 단계로 나누어 수행된다. 스케쥴링의 첫 번째 단계는 태스크 그래프에서 병렬성이 있는 태스크들을 선정하여 한 번에 스케쥴링할 태스크들의 그룹을 찾아내는 단계로, 이 단계를 수행하면 태스크 그래프상의 모든 태스크들이 동일한 시간 대역내에 독립적으로 할당할 수 있는 태스크들로 그룹을 이루게 된다. 두 번째 단계에서는 각 태스크를 수행할 머신을 선정한다. 머신을 할당하기 위한 태스크의 순서는 첫 번째 단계에서 결정된 태스크 그룹의 우선순위를 따르며 같은 그룹에 속한 태스크간의 순서는 태스크의 크기순으로 한다. 각 태스크는 머신들 중 태스크의 수행을 가장 빠르게 완료할 수 있는 머신에 할당한다. 이를 위해서는 각 머신상에서의 태스크 수행완료시간을 계산해야 하며, 태스크 수행완료시간은 각 부모노드로부터의 자료전송시간과 머신상에서의 태스크 수행시간으로부터 계산될 수 있다. 이 때, 다른 머신에 할당된 부모노드로부터의 자료 전송이 매우 긴 경우에는 자료를 전송받는 대신 부모노드를 복사하여 직접 수행하므로써 태스크의 수행시간을 앞당길 수 있다. 따라서 이와 같은 부모노드의

복사 여부와 이기종 머신간에 서로 다른 태스크 수행 시간을 감안하여 머신을 선정한다.

기존의 태스크 복사기법들[18,26]에서는 모든 부모노드 뿐 아니라 부모노드 이상의 상위 노드들에까지 재귀적으로 복사여부를 확인하므로 스케줄링 알고리즘의 복잡도가 매우 높으며, 태스크의 수가 증가할수록 실제적인 사용이 불가능해진다. 이러한 단점을 극복하기 위하여 본 논문에서는 태스크의 수행을 가장 늦추는 태스크로 복사의 대상을 제한하여 가용한 알고리즘이 되도록 복잡도를 유지하였다. 다양한 태스크 그래프에 대한 실험 결과는 이와 같은 부분적인 태스크의 복사로도 응용 프로그램의 전체 수행시간이 현저히 단축됨을 보여준다.

본 논문의 구성은 다음과 같다. 제 2 절에서는 정적 스케줄링에 관한 기존의 연구들을 개관하고 이들을 분석하여 본 연구에서 제안한 방법과의 차이점을 기술하였다. 또한, 제 3 절에서는 이기종 스케줄링 기법을 제안하고 알고리즘을 분석하였다. 제 4 절에서는 다양한 태스크 그래프를 대상으로 기존의 스케줄링 기법과 본 논문에서 제안한 기법을 모의실험하고 그 결과를 비교하였다. 마지막으로 제 5 절에서는 본 연구의 결론과 앞으로의 연구방향을 수록하였다.

2. 관련 연구

분산환경을 위한 정적 스케줄링에 관한 연구는 경험적인 방법(heuristic methods)이 주를 이루어 왔다. 그 중의 한가지 방법이 각 태스크의 수행속도가 가장 빠른 머신을 선정하여 태스크를 할당하는 방법이다[18]. 이 기법은 통신비용을 고려하지 않기 때문에 알고리즘의 복잡도가 $O(nq)$로 매우 간단하다. 여기서 n과 q는 각각 태스크와 머신의 총 수를 의미한다. 반면 상이한 머신에 할당되는 태스크간의 자료 의존성을 무시하므로 태스크의 할당 결과가 통신시간을 과도히 요구하는 결과를 초래할 우려가 있다. 특히, 머신간의 성능 차이가 적은 경우에는 많은 수의 머신을 사용할수록 통신으로 인하여 전체적으로 작업의 종료시간이 늦어지는 반작용이 발생할 수 있다. 그러나 통신비용이 태스크의 계산 시간에 비하여 매우 작은 환경에서는 매우 좋은 결과를 얻을 수 있다[18].

Iverson의 리스트 스케줄링 기법[16,17]은 태스크의 시작시간을 위주로 먼저 작업을 수행할 수 있는 태스크들을 우선적으로 머신에 할당하는 기법이다. 즉, 어떤 태스크를 머신에 할당하기 위해서는 이 태스크가 필요로 하는 자료를 생성하는 태스크들이 모두 종료되었다고 가정한다. 여기서 만일 자료의존성을 지니는 태스크가 같은 머신에 할당되어 있다면 자료전송을 위한 시간은 없는 것으로 간주하며, 이와 반대로 다른 머신에 할당된 경우에는 네트워크를 통하여 계산 결과를 전달받아야 하므로 시작시간이 늦어지게 된다. 따라서 리스트 스케줄링 기법에서는 계산에 필요한 모든 자료를 전송 받아서 태스크가 수행을 시작할 수 있는 시점을 위주로 할당 여부를 결정한다. 또한 같은 우선순위를 지니는 태스크의 수가 머신의 수보다 많은 경우, 이들을 그룹화하여 머신의 수와 동일한 그룹을 형성하고 하나의 그룹이 할당된 머신은 나머지 그룹을 할당할 수 없도록 하였다. 이러한 가정은 자료의존성이 있는 태스크들을 서로 다른 머신으로 분산시키므로써 통신오버헤드를 발생시키는 원인이 된다. 이기종 환경에서의 리스트 스케줄링 기법에서는 두 대 이상의 머신에서 수행시작 시간이 같더라도 태스크를 수행하는데 걸리는 시간이 머신마다 다르므로 태스크의 수행완료시간이 각각 다를 수 있다는 가능성을 고려하여 태스크의 수행완료 시점을 비교하여 작업의 크기가 큰 태스크가 가장 빨리 종료될 수 있는 머신에 우선 할당하는 전략을 이용한다. 이 기법의 복잡도는 $O(n^2q)$ 이다.

Tao[12]의 반복 할당 기법은 simulated annealing에 기초한 태스크 할당 기법이다. 반복 할당 기법은 스케줄링에 의한 태스크 수행완료시간이 프로그래머가 지정한 기준을 만족할 때까지 스케줄링을 반복한다. 즉, 임의의 할당함수에 의하여 모든 태스크들을 스케줄링하고 그 결과에 대하여 만족도를 계산한다. 여기서, 스케줄링 결과의 만족도는 각 태스크의 수행완료 시간이 미리 결정된 시간내에 처리된 비율, 또는 태스크 그래프의 전체 수행시간이 주어진 전체 수행시간의 상한을 만족하는지의 여부 등을 의미한다. 이와 같이 스케줄링 결과의 만족도를 계산하고 이 값이 조건을 만족하지 않는 경우에는 이전단계에서 스케줄링 된 태스크의 스케줄링 결과를 입력으로 하여 임의의 할당함수를 재적용한다. 이러한 스케줄링 단계는 프로그래머가 미리 지정한 조건을 만족할 때까지 반복된다. 이 방법은 각 태스크를 할당하는 할당함수에 따라 성능의 차이를 보일 수 있으며 프로그래머가 만족될 수 없는 조건을 제시하는 경우에는 스케줄링이 무한히 수

행될 수 있다.

Lo의 전역 최적화 (global optimization) 기법[13]은 사용되는 머신과 태스크를 노드로 표현하고, 태스크간의 자료의존성과 태스크가 노드에 할당될 가능성을 간선으로 나타낸 네트워크 그래프에서 min cut[13] 알고리즘을 이용하여 태스크와 머신을 그룹화하는 방법이다. Min-cut 알고리즘에 의하여 그룹화 된 노드들이 모두 태스크인 경우는 태스크들을 하나의 태스크로 병합하는 클러스터링을 의미하며, 노드들이 태스크와 머신의 집합인 경우는 그룹내의 태스크들이 같은 그룹에 속한 머신에 할당됨을 의미한다. 이와 같은 전역 최적화 기법은 알고리즘이 간단하며 실용적이다. 그러나 전역 최적화 기법에 의하여 얻어지는 스케쥴링 결과는 태스크의 전체 수행완료 시간을 줄이기보다는 각 머신상에서 수행되는 태스크들의 수행시간과 통신시간의 총합을 최소화한다. 그러므로 태스크를 클러스터로 묶는 과정에서 통신시간을 줄이기 위하여 병렬성을 희생시킬 수 있으며, 경우에 따라서는 대부분의 태스크들이 한 머신에 할당되므로 인하여 응용 프로그램의 수행완료시간이 매우 길어지는 반작용이 발생할 수 있다.

과도한 통신이 발생하는 리스트 스케쥴링의 단점을 보완하기 위하여 통신의 영향을 줄일 수 있는 경우에 한하여 태스크를 여러 머신에 복사할 수 있도록 허용하는 복사기법도 연구되었다[18]. 이 방법은 통신시간이 과도히 늘어나는 경우에 이러한 결과를 유발하는 태스크를 머신에서 직접 실행하여 필요한 결과를 얻도록 함으로써 통신 비용이 큰 시스템의 경우에 발생하는 반작용을 원인적으로 제거할 수 있는 장점이 있다. Kruatrachue[18]는 이 기법을 통신비용이 작은 다중 프로세서 시스템에 적용하여 명령어 단위의 병렬처리를 시도하였다. 그러나 자료 의존관계에 있는 모든 태스크들에 대하여 복사를 허용하기 때문에 알고리즘의 복잡도가 매우 높으므로 분산환경에서는 실용적이지 못한 단점이 있다. Manoharan[26]은 복사기법의 성능을 보다 향상시키기 위하여 재귀적인 태스크 복사기법을 제안하였다. 이 기법은 응용 프로그램의 수행시간을 앞당길 수 있으나 Kruatrachue의 연구와 마찬가지로 복잡도가 $O(n^3q)$으로 매우 커서 실제적으로 사용하기 어려운 단점이 있다. 본 논문에서는 복사기법을 이기종 환경에 맞도록 보완하고 복잡도를 $O(n^2q)$로 유지하는 알고리즘을 제안하였다.

3. 이기종 태스크 스케쥴링

응용 프로그램의 병렬처리를 위해서는 프로그램이 작은 단위의 태스크들로 분할되고, 태스크간의 자료 의존관계는 비순환 태스크 그래프 $G=(V,E)$로 표현된다. 여기서 $V=\{v_1, v_2, \cdots, v_n\}$는 n개의 태스크로 구성된 태스크 집합이며, E는 자료 의존관계가 있는 태스크들간의 유향간선(directed edge)의 집합이다. 태스크 v_i에서 v_j의 방향으로 자료 의존성이 존재할 때, v_i와 v_j간의 유향간선을 e_{ij} ($e_{ij}\in E$)로 나타낸다. 즉, $e_{ij}\neq 0$ 이면, 태스크 v_i와 v_j간에 자료 의존성이 존재하며 태스크 v_i는 태스크 v_j에 앞서 수행되어야함을 의미한다.

$P=\{p_1, p_2, \ldots p_q\}$는 q개의 머신으로 구성된 이기종 시스템을 의미한다. $X_{n\times q}$는 각 태스크들이 서로 다른 머신들에서 수행될 때의 수행시간을 나타내는 배열이며 $X_{n\times q}$의 원소 x_{ij}는 머신 p_j상에서 태스크 v_i를 수행할 때 소요되는 시간, 즉 노드의 크기이다. $C_{n\times n}$는 태스크들간의 통신배열이며 원소 c_{ij}는 태스크 v_i로부터 v_j로 전송되는 자료의 바이트 수이다.

$\mu : V\rightarrow P$ 는 할당 함수로서 태스크 v_i를 머신 p_k에 할당하는 경우 $\mu(i)=k$이다.

$\delta(c_{ij}, p_k, p_r)$는 c_{ij} 바이트의 자료를 p_k에서 p_r로 전송하기 위한 통신시간을 나타낸다. 여기서, $\delta(c_{ij}, p_k, p_r)$는 머신 p_k와 p_r의 자료 저장형식이나 표현 형식이 다른 경우에는 이를 변환하기 위한 오버헤드를 포함한 통신시간으로 간주한다. PVM(Parallel Virtual Machine)[24] 환경에서도 자료의 변환비용을 통신비용에 포함하고 있으며 본 논문에서 고려하는 태스크 그래프 모델에서도 $\delta(c_{ij}, p_k, p_r)$을 자료 변환 및 전송에 소요되는 시간의 합으로 간주한다.

3.1 병렬 태스크 그룹화

리스트 스케쥴링 기법[16,17]이나 전역 최적화 기법[13]등에서 연구된 바와 같이 스케쥴링 되는 태스크들의 순서는 스케쥴링의 성능에 영향을 미칠 수 있다. 동일한 스케쥴링 기법을 적용하는 경우에도 스케쥴링 되는 태스크들의 순서에 따라 서로 다른 결과가 산출될 수 있다. 본 논문에서는 태스크들의 순서 결정을 위

하여 2단계 접근 방식을 이용한다.

먼저 태스크간의 자료 의존성에 기초하여 병렬로 수행될 수 있는 태스크들에게 동일한 우선순위를 부여한다[17]. 이를 구현하기 위해서는 태스크 그래프의 뿌리 노드(root node)에 가장 높은 우선 순위(0)를 부여하고 뿌리 노드로부터 각 노드에 이르는 경로의 최대 길이를 각 노드의 우선 순위로 삼아 우선 순위가 동일한 태스크들을 병렬 태스크로 구분한다. 즉, 뿌리 노드에 속하는 모든 태스크 v_i의 l_i을 0 이라고 하고 나머지 태스크 v_i의 우선 순위를

$$l_i = \max\{l_j | \text{for } \forall v_j \text{ such that } e_{ji} \neq 0\} + 1$$

로 결정한다. 이러한 우선순위 부여 방식에 따라 자료 의존관계로 인하여 태스크의 수행이 지연되는 현상을 방지할 있다.

같은 우선순위를 부여받은 태스크들의 스케쥴링 순서에 따라서도 응용 프로그램의 수행시간이 영향을 받을 수 있다. 본 논문에서는 같은 우선순위의 태스크들에 대하여 크기가 큰 태스크를 먼저 스케쥴링하도록 하므로써 수행시간이 단축될 수 있는 가능성을 높이도록 하였다. 각 태스크에 대한 수행시간이 머신에 따라 다르므로 태스크의 크기를 비교하기 위해서는 모든 머신에서의 수행시간에 대한 평균값을 사용한다.

3.2 태스크 복사를 이용한 머신 할당

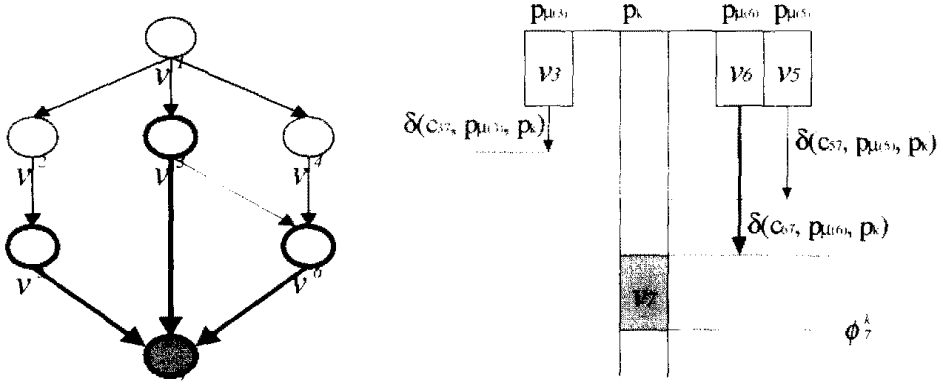
이기종 환경에서는 하나의 태스크를 여러 머신에서 수행시킬 때 각 머신에서의 태스크 수행시간이 다르다는 점을 고려해야 한다. 즉, 여러 머신에서 태스크의 수행이 시작될 수 있는 시점이 같더라도 수행이 완료되는 시점은 각기 다르다. 그러므로 태스크의 수행완료 시간을 기준으로 가장 빠르게 수행을 완료할 수 있는 머신을 할당한다.

이기종 환경상에서 태스크 복사 기법의 기본 개념은 어떤 태스크에 머신을 할당할 때,

① 자료 의존관계에 있는 태스크들이 다른 머신에 할당된 경우, 이들로부터 통신을 통하여 자료를 전달 받는 경우의 태스크 수행완료 시간과

② 통신에 의한 자료를 전달 받기 위한 머신의 유휴상태 대신 필요한 결과를 산출하는 태스크를 복사하여 중복 수행하도록 할 경우의 태스크 수행완료 시간을 비교하여 ②의 결과가 ①에 비하여 나은 경우에 태스크의 복사를 허용하도록 하는 것이다.

태스크 v_i를 어떤 머신 p_k에 할당하려고 할 때, 태스크 v_i의 수행이 시작될 수 있는 시점은 v_i와 자료 의존관계에 있는 모든 노드들의 수행이 완료된 시점으로부터 통신을 통하여 모든 노드들의 결과를 전달받은 시점이 된다. 이 때, v_i와 자료 의존관계에 있는 임의의 노드 v_j가 v_i와 같은 머신에 할당되어 있다면, 즉 $\mu(j) = k$ 이면 두 노드간에는 통신이 없는 것으로 간주할 수 있다. 따라서 v_i의 수행시작 시점은 v_i와 자료 의존관계에 있는 노드들 중 $\mu(j) \neq k$인 노드들로부터의 자료 전송만을 고려한다. 예를 들어, (그림 1a)에서 v_7의 수행이 시작될 수 있는 시점은 (그림 1b)에서와 같이 v_6의 수행이 완료되고 v_6으로부터 v_7로의 자료전송이 끝나는 시점이다. 머신 p_k에 할당된 태



(그림 1) 자료 의존관계의 영향
(Fig. 1) Effects of dependencies

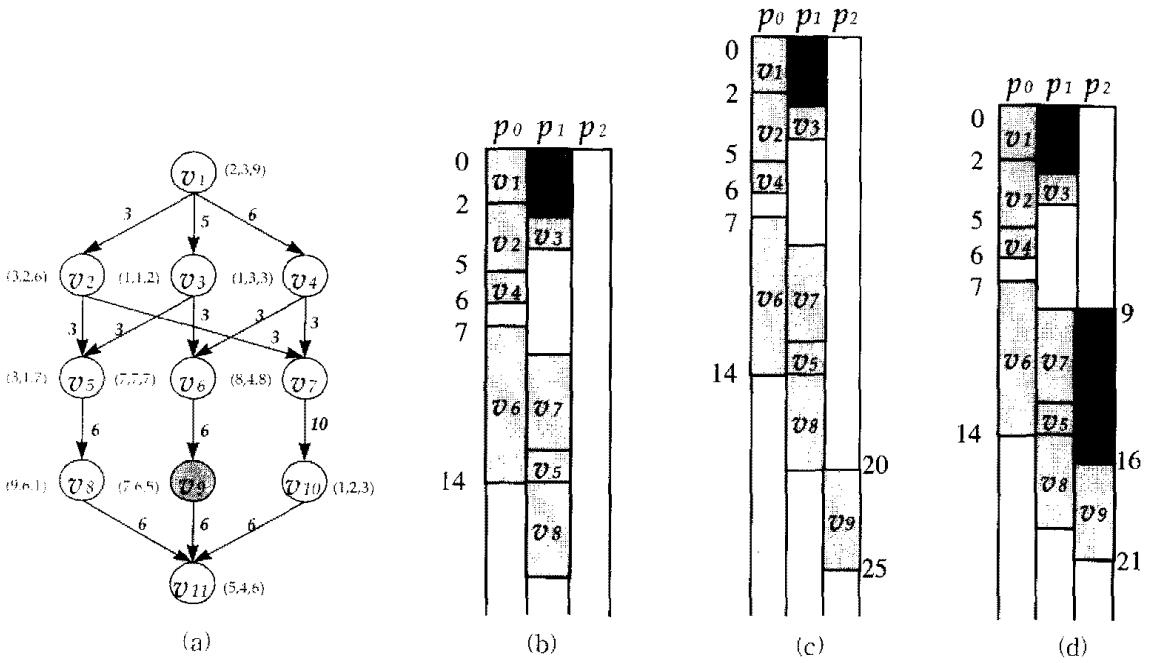
[illegible]

v_f의 수행이 완료되는 시점을 $\phi_f^{\mu(f)}$라 하고 태스크 v_i의 모든 선행노드들의 집합을 G_i라 할 때, 태스크 v_i가 자료 의존관계에 있는 태스크들로부터 통신을 이용하여 결과를 전달받는 경우의 수행시작 시점은 $\max_{v_f \in G_i} \{\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k)\}$로 정의할 수 있으며 수행완료시간은 $\max_{v_f \in G_i} \{\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k)\} + x_{ik}$이다.

G_i에 속하고 p_k가 아닌 다른 머신에 할당된 태스크 중 v_i의 수행시작을 가장 늦추는 태스크를 v_f라 할 때, $\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k) = \max_{v_f \in G_i \text{ and } \mu(f) \neq k} \{\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k)\} + x_{ik}$이다. 머신 p_k에 할당된 모든 노드들의 수행이 완료되는 시점을 Φ_k라 하고 $\Phi_k \geq (\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k))$라면 태스크 v_i의 수행은 Φ_k에서 시작되어야 하며 따라서 수행종료시간은 $\Phi_k + x_{ik}$이다. 이 경우에는 통신시간의 단축이 수행시간의 단축에 영향을 미치지 않는다.

반면 태스크 v_f의 수행완료시간이 매우 늦거나 v_f로부터의 통신시간이 커서 머신 p_k에서 태스크 v_i의 수행시작 시점을 지연시키는 경우에는 통신을 통하여 v_f의 결과를 전달받는 대신 v_f를 복사하여 머신 p_k상에서 수행하므로써 수행시작 시점을 앞당기는 효과를 얻을 수 있다. 이 때, p_k상에서 v_f의 수행은 v_f와 자료 의존관계에 있는 태스크 v_w중에서 $\mu(w) \neq k$인 태스크들로부터의 통신이 완료된 시점에서 시작될 수 있다. 즉, 태스크 v_f는 $\max\left\{\Phi_k, \max_{v_w \in G_i} \{\phi_w^{\mu(w)} + \delta(c_{wf}, p_{\mu(w)}, p_k)\}\right\}$의 시점에서 시작될 수 있으며 이 경우 태스크 v_i의 수행시작 시점은 $\max\left\{\Phi_k, \max_{v_w \in G_i} \{\phi_w^{\mu(w)} + \delta(c_{wf}, p_{\mu(w)}, p_k)\}\right\} + x_{ik}$, 수행완료 시점은 $\max\left\{\Phi_k, \max_{v_w \in G_i} \{\phi_w^{\mu(w)} + \delta(c_{wf}, p_{\mu(w)}, p_k)\}\right\} + x_{ik}$이다.

예를 들어, (그림 2b)는 (그림 2a)의 태스크 그래프에 대한 스케쥴링 과정 중 v_8까지의 태스크들이 스케쥴된 결과이다. (그림 2a)에서 괄호안의 숫자는 태스크가 각 머신에서 수행될 경우의 수행시간을 의미하며 간선상의 숫자는 간선으로 연결된 태스크간의 통신시간을 의미한다. v_9를 머신 p_2에 할당하기 위해서는 v_6으로부터의 결과가 필요하며, 통신을 통하여 v_6의 결과를 전송받아 v_9를 시작할 수 있는 시점은 (그림 2c)와 같이 시점 20 이다. 그러나 머신 p_2에 v_6를 복사하여 수행하는 경우에는, v_6의 수행이 시점 9에서 시작하여 16에 종료되므로 v_9의 시작시점이 16으로



(그림 2) 태스크 복사에 의한 태스크 스케쥴링
(Fig. 2) Task scheduling using task duplication

앞당겨 진다. 따라서 (그림 2d)와 같이 v_k는 p_2로 복사하도록 결정한다.

어떤 태스크 v_i에 머신을 할당하기 위해서는 이기종 시스템을 구성하는 모든 머신에 대하여 통신을 통해 자료를 전달받는 경우와 태스크를 복사하여 수행하는 경우의 수행 종료시간을 계산하고 이들 중 v_i의 수행을 가장 빠르게 완료할 수 있는 머신을 선정한다. 만일 태스크 복사에 의한 스케줄링이 가장 빠른 수행 완료시간을 산출하는 경우에는 선택된 머신에 복사가 요구되는 태스크들을 복사한 후 v_i를 할당한다. (그림 3)은 위에서 설명한 바와 같이 태스크 복사를 이용한 머신 할당 기법을 보여준다.

Program Heterogeneous Duplication Scheduling
input : Task set $V = \{v_1, v_2, \cdots, v_n\}$
Edge set $E = \{e_{ij} \mid 1 \le i, j \le n\}$
Machine Set $P = \{p_1, p_2, \cdots, p_q\}$
output : Task allocation

BEGIN
Initialize job queues M_a, $a = 1,2, \cdots q$ to be empty for each machine.
for each task v_j, $j = 1,2, \cdots n$ **do**
  Determine the level of each task $l_i = \max\{l_k \mid \text{for} \forall v_k, e_{kj} \neq 0\} + 1$
end for
Let L_m **be the set of tasks** v_j **such that** $l_j = m$.
for each group L_m **do**
  Sort the tasks $v_b \in L_m$ in decreasing order of $\overline{x_b}$.
  // $\overline{x_b}$ be the average execution time of tasks.
  Call HDS_MAP(L_m)
  end for
END

Procedure HDS_MAP(L_m)
BEGIN
for each task $v_i \in L_m, 1 \le i \le |L_m|$ **do**
  Initialize set B_k, $k = 1,2, \cdots q$ to be empty.
  for each machine p_k, $k = 1,2, \cdots q$ do
  // let ϕ_i^k be the completion time of v_i at p_k.
  // let Φ_k be the maximum of the completion times of tasks allocated to p_k so far.
    Find a task $v_f \in G_i$ which maximizes $\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k)$;
    if $\Phi_k \ge (\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k))$ **then** // duplicating v_f at p_k is not helpful.
      Add task v_i to B_k ;
      $t_k = \Phi_k + x_{ik}$;
   else
    // test if duplicating v_f at p_k can shorten the completion time.
    Find a task $v_w \in G_f$ which maximizes $\phi_w^{\mu(w)} + \delta(c_{wf}, p_{\mu(w)}, p_k)$;
    if $\left(\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k)\right) \ge \left(\max\left\{\Phi_k, \left(\phi_w^{\mu(w)} + \delta(c_{wf}, p_{\mu(w)}, p_k)\right)\right\} + x_{fk}\right)$ **then**
      Add tasks v_f to B_k;
      $t_k = \max\left\{\Phi_k, \left(\phi_w^{\mu(w)} + \delta(c_{wf}, p_{\mu(w)}, p_k)\right)\right\} + x_{fk} + x_{ik}$;
    else // duplicating v_f at p_k turns out to be not helpful.
      $t_k = \left(\phi_f^{\mu(f)} + \delta(c_{fi}, p_{\mu(f)}, p_k)\right) + x_{ik}$;
    end if
    Add task v_i to B_k ;

  end if
 end for
 Find a machine p_a which minimizes t_a;
 Add the tasks in B_a to M_a;
end for

END

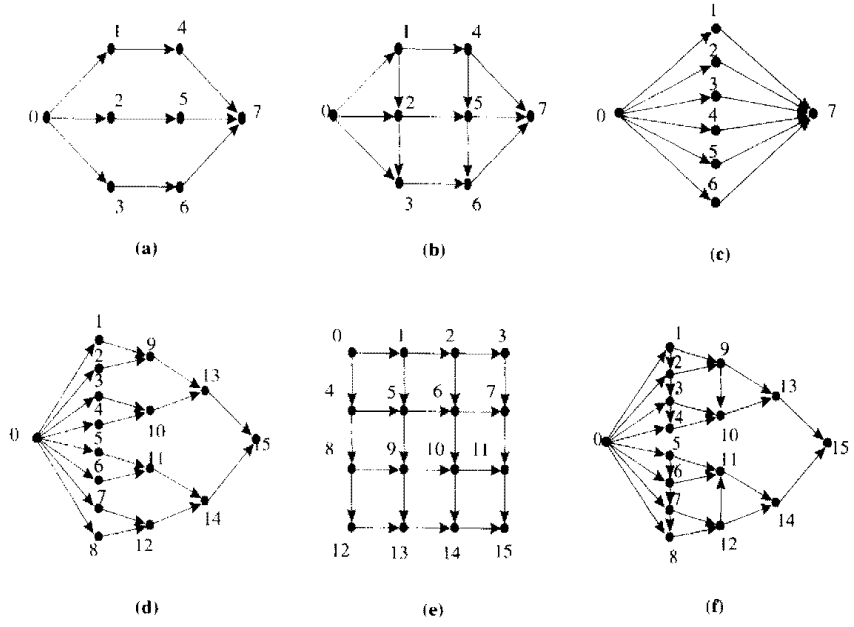
(그림 3) 이기종 복사 스케줄링 알고리즘
(Fig. 3) Heterogeneous Duplication Scheduling Algorithm

(그림 3)에서 제시한 알고리즘의 시간 복잡도(time complexity)는 다음과 같다. 태스크 그래프내의 모든 태스크 v_i에 대하여 v_i가 각 머신 p_k에 할당되었을 때의 수행종료시간 q를 계산하기 위하여 $v_j, e_{ji} \in E$ 중 v_i의 수행을 가장 지연시키는 태스크 v_j를 찾아야 하고 v_j를 p_k에 복사할지 여부를 결정하기 위하여 $v_m(e_{mi} \in E)$ 중 v_i의 수행을 가장 지연시키는 태스크를 찾아야 한다. v_j와 v_m의 개수는 각각 indegree(v_i)와 indegree(v_j)이므로 그래프내의 모든 태스크들을 할당할 때 검색되는 v_j와 v_m의 개수의 합은 $2E$, 즉 $O(|E|)$이다. 여기서, $|E|$의 최대값은 $|V|^2$이다. 태스크 하나에 대하여 각 머신 p_k에 할당되었을때의 수행종료시간을 계산하게 되므로 전체 시간 복잡도는 $O((|V|+|E|)q)$, 즉 $O((|V|+|V|^2)q) \equiv O(|V|^2 q)$로 나타낼 수 있다.

4. 실험 및 고찰

본 논문에서 제안한 HDS(Heterogeneous Duplication Scheduling) 기법의 성능분석을 위하여 다양한 유형의 태스크 그래프에 대한 모의실험을 수행하였다. HDS 기법의 성능 비교를 위하여 2장에서 설명한 스케쥴링 기법 중 통신을 고려하지 않는 기법과 리스트 스케쥴링 기법의 결과와 비교하였다. 본 논문에서는 통신을 고려하지 않는 스케쥴링 기법을 THS(Traditional Heterogeneous Scheduling) 기법이라 하고 이기종 리스트 스케쥴링 기법을 HLS(Heterogeneous List Scheduling)이라고 명명하였다.

첫 번째 실험에서는 (그림 4)에 주어진 다양한 형태의 그래프들에 대하여 스케쥴링을 수행하였다. (그림 4)에 보인 6가지 태스크 그래프들은 실제 응용 프로그램에서 나타날 수 있는 형태들로 (그림 4)의 (a), (c) 및 (d)는 데이터가 여러 머신들에 분산되어 작업이 수행되고 그 결과가 다시 하나의 머신으로 모이는 형태의 그래프이다. 또한 (그림 4)의 (b), (e) 및 (f)는 서로 이웃한 머신들간에 통신을 유발시키면서 작업이 수행되는 형태이다. (그림 4)의 (a)-(c)는 2 머신, 그리고 (그림 4)의 (d)-(f)는 4 머신으로 구성된 이기종 환경에서 수행된다고 가정하였다. 이기종 시스템은 다양한 성능과 특성의 머신으로 구성될 수 있다. 따라서 이들 시스템의 성능을 모의하기 위하여 이기종 시스템을 구성하는 머신의 통신성능과 연산성능의 비율(연산/통신의 성능비)에 따라 각 노드의 크기 및 노드간의 통신시간을 임의로 발생시키면서 스케쥴링 결과를 비교하였다. 이 때, 연산/통신의 성능비는 시스템에 따라 다양한 값으로 나타나게 된다. 예를 들어, 분산메모리 MIMD 머신의 경우에는 배정도 실수에 대한 연산/통

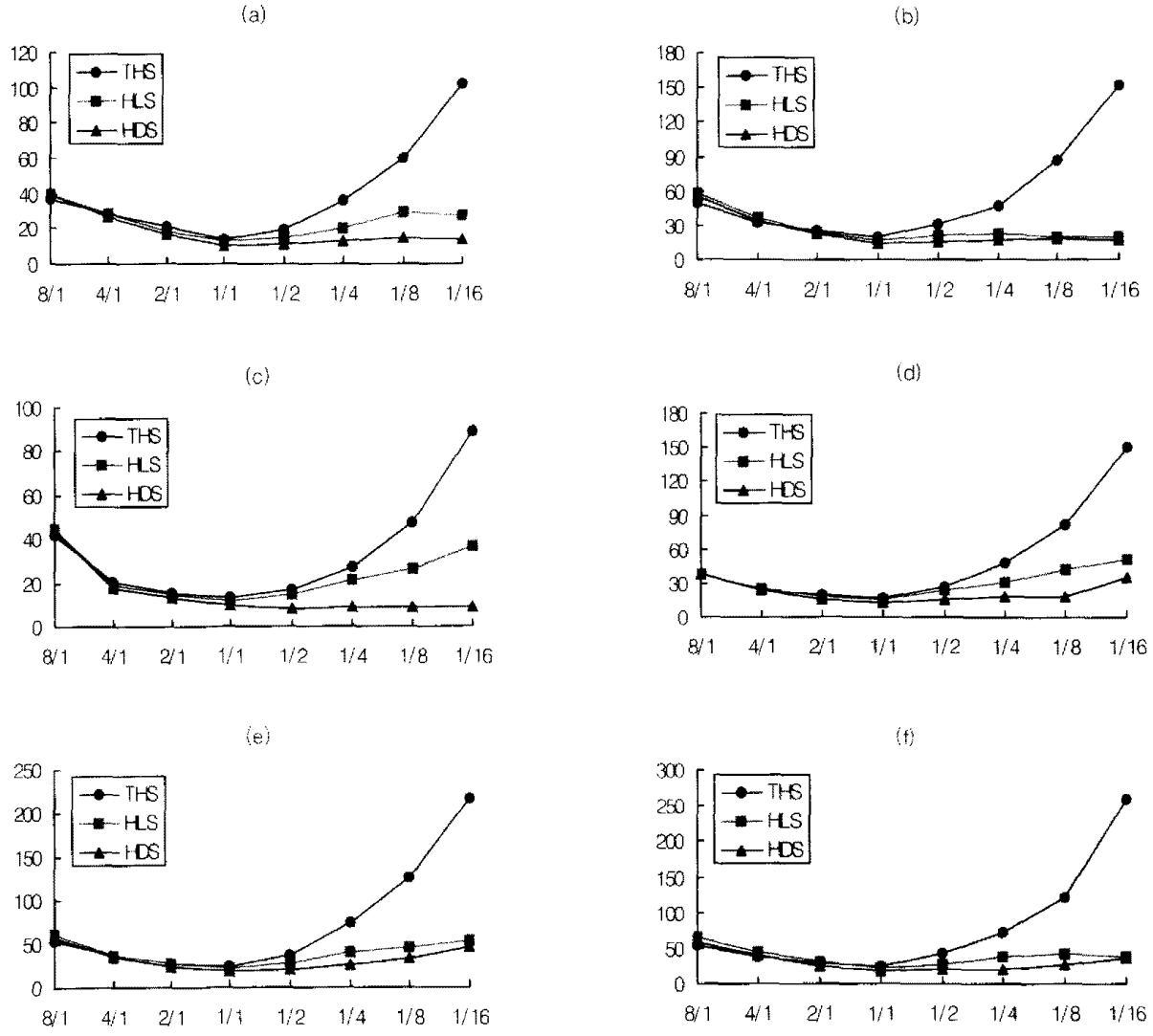


(그림 4) 여러 가지 태스크 그래프
(Fig. 4) Task graphs

신의 성능비가 1:10~1:20 수준이며 공유메모리 MIMD에서는 연산시간에 비하여 통신시간이 무시 할 정도로 작다[21]. 따라서 본 논문에서는 연산/통신의 성능비를 8:1에서 1:16까지 변화시키면서 이 비율의 상한비가 만족되는 노드와 노드간의 유향간선의 크기를 임의로 발생시켰다. 따라서 이 값은 공유메모리 MIMD로부터 분산환경에 이르는 다양한 환경을 포함한다. 또한 모의실험의 결과가 특정한 경우에 의하여 결정되는 것을 방지하기 위하여 동일한 조건하에서 10회씩 임의의 값을 발생시켜 스케쥴링을 수행하고 평균값으로 스케쥴링의 결과를 제시하였다.

(그림 5)는 (그림 4)의 그래프 집합에 대한 스케쥴링 모의실험 결과를 보여준다. (그림 5)에서 x 축은 연산/통신의 성능비를 나타내며, y 축은 수행 완료시간을 나타낸다. (그림 5)의 결과에서 연산/통신 성능비가 8:1인 경우에는 그래프의 종류에 관계없이 THS에 의한 스케쥴링 결과가 나머지 방법들에 비하여 우수함을 알 수 있다. 이것은 THS가 통신을 전혀 고려하지 않은 스케쥴링 기법이므로 통신시간이 매우 작은 경우에 최적의 결과를 나타내기 때문이다. 따라서 통신시간의 비율이 적어질수록 THS가 최적에 근접한 결과를 보여주고 있다.

연산/통신의 성능비가 작아지는 경우에는 모든 태스크 그래프에 대하여 HDS가 HLS나 THS에 비하여 우수함을 알 수 있다. 본 논문에서 다루는 이기종 컴퓨팅 환경에서는 네트워크로 연결된 머신들간의 통신이

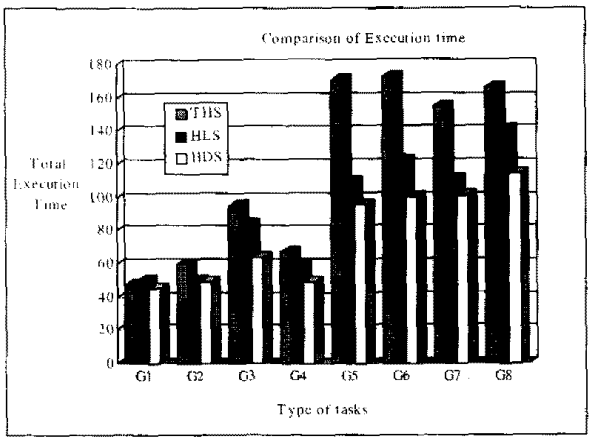


(그림 5) (그림 4)의 태스크 그래프별 실험 결과
(Fig. 5) Execution time for task graphs in figure 4

성능차이며, 연산/통신의 성능비가 작으므로 이기종 컴퓨팅 환경에서는 HDS 기법이 기존의 스케쥴링 기법에 비하여 우수하다고 할 수 있다. 특히 연산/통신의 성능비가 매우 작으며 태스크간의 병렬성이 낮은 경우, 예를 들면 (그림 5)의 (b) 또는 (f)에서 연산/통신의 성능비가 1:16 이하인 경우에는 HLS와 HDS의 결과가 유사하게 나타나는 것을 볼 수 있는데 이는 적은 수의 머신에 태스크들이 할당되기 때문이며 최악의 경우에는 하나의 머신에만 태스크가 할당되기 때문이다. 따라서 HDS는 HLS에 비하여 나쁜 결과를 초래할 수 없다.

태스크 그래프에 대한 스케쥴링은 응용 프로그램의 태스크 그래프가 어떠한 형태인가에 의해서도 영향을 받는다. 보다 다양한 형태의 태스크 그래프에 대한 스케쥴링 성능을 분석하기 위하여 두 번째 실험에서는 임의의 모양을 가진 그래프들을 발생시켜 실험하였다. 이 실험을 위하여 그래프의 노드 수가 8개 또는 19개일 때 생성될 수 있는 임의의 모양을 생성하였다. 생성된 그래프의 모양은 뿌리노드와 잎노드가 각각 하나씩이고 이들을 제외한 중간단계의 노드가 8개 또는 19개인 형태이다.

각각의 태스크 그래프는 각각 4 머신으로 구성된 이기종 환경에서 수행되는 것으로 가정하였다. 다양한 형태의 그래프 생성을 위해서 완전결합 그래프의 유향간선의 수를 100%라고 기준할 때, 유향간선의 수를 30%에서 60%까지 변화시키면서 그래프를 생성하였고 연산/통신의 성능비를 8:1에서 1:16까지 변화시키면서 세 가지 스케쥴링에 의한 스케쥴링 결과를 비교하였다. (그림 6)은 각각의 스케쥴링 기법을 적용하였을 경우의 수행 완료시간에 대한 평균값을 나타낸 것이다. $G_1 \sim G_4$는 노드가 8개인 경우, $G_5 \sim G_8$는 노드가 19개인 경우의 그래프에 대한 실험 결과이다. (그림 6)에서도 알 수 있듯이 임의의 그래프 집합에 대해서도 HDS가 기존의 스케쥴링 기법들에 비하여 이기종 환경에 적합함을 보여준다.



(그림 6) 임의 발생된 태스크 그래프에 대한 실험 결과
(Fig. 6) Execution time for randomly generated task graphs

5. 결 론

이기종 분산처리의 목적은 태스크들을 병렬처리 특성에 알맞은 여러 가지 이기종 머신에 할당하여 응용 프로그램의 전체 수행 시간을 단축하는 것이다. 이를 위해서는 프로그램을 여러 개의 태스크로 분할하고 각각의 태스크를 적합한 머신에 할당하는 스케쥴링이 매우 중요하다. 동일기종 환경과 달리 이기종 환경에서는 태스크의 수행시간이 머신의 특성에 따라 다르며 자료의 전송시 자료 저장형식을 변환하기 위한 오버헤드를 고려해야 한다. 특히, 이기종 환경에서도 병렬성을 높이면 통신시간이 증가하고, 반대로 통신시간을 줄이려면 병렬성이 줄어드는 반작용을 피할 수 없다. 본 논문에서는 두 태스크간에 과다한 통신이 발생할 때, 두 개의 태스크를 통합하거나 통신을 이용하여 자료를 전송 받기보다는 필요한 태스크를 여러 머신에서 동시에 수행하도록 함으로써 통신으로 인한 지연 시간을 줄이고 병렬성을 훼손하지 않는 이기종 복사 스케쥴링 기법을 제안하였다. 본 논문에서 제안된 이기종 복사스케쥴링에서는 어떤 태스크와 자료 의존관계가 있는 태스크들 중 그 태스크의 수행시작 시점을 가장 지연시키는 태스크에게만 복사를 허용하므로써 알고리즘의 복잡도를 줄이는 동시에 통신으로 인하여 발생하는 전체 수행시간을 줄이도록 하였다.

제안한 기법과 기존의 이기종 스케쥴링 기법을 여러 가지 다양한 태스크 그래프들에 적용하여 모의실험한 결과 본 논문에서 제안한 복사 스케쥴링 기법이 기존의 스케쥴링 기법들에 비하여 우수한 결과를 보여주고 있음을 알 수 있었다. 특히, 단위 태스크의 수행 시간에 비하여 통신시간의 비율이 큰 응용프로그램의 이기종 연산에 있어서 본 논문에서 제안한 기법이 매우 우수한 성능을 발휘함을 알 수 있다.

참 고 문 헌

[1] M. H. Willebeek LeMair, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Trans. Parallel Distributed Systems*, Vol.4, No.9, pp.979-993, Sep. 1993.

[2] K. H. Shum and K. Moody, "A competitive environment for parallel applications on heterogeneous workstation clusters," *in Proc. 1996 Workshop on Heterogeneous processing*, 1996.

[3] A. Bricker, M. Litzkow and M. Livny, "Condor technical summary," Technical Report, CS TR-92-1069, 1992.

[4] T. Y. Suen and S. K. Wong, "Efficient task migration algorithm for distributed systems," *IEEE Trans. Parallel Distributed Systems*, Vol.3, No.4, pp.488-499, Jul. 1992.

[5] J. B. Weissman, "The interface paradigm for network job scheduling," *in Proc. 1996 Workshop on Heterogeneous processing*, 1996.

[6] J. Casas, R. Konuru, S. W. Otto, "Adaptive load migration systems for PVM," *in Proc. Supercomputing*, 1994.

[7] J. Prouty, S. Otto and J. Walpole, "Adaptive execution of data parallel computations on networks of heterogeneous workstations," Technical Report, CSE-94-012, Mar. 1994.

[8] R. Thakur, A. Choudhary and J. Ramanujam, "Efficienct algorithms for array redistribution," *IEEE Trans. Parallel Distributed Systems*, Vol.7, No.6, pp.587 594, Jun. 1996.

[9] D. Coppit, D. Engler, S. McCulloch, "The distributed java othello environment," http://www.cs.virginia.edu/~dae4e/java-othello/writeup.html

[10] A. Wollrath, J. Waldo and R. Riggs, "Java-centric distributed computing," *IEEE MICRO*, Vol.17, No.3, pp.44-53, Jun. 1997.

[11] M. J. Quinn, *Parallel Computing : Theory and Practice*, McGraw-Hill Book Company, 1993.

[12] L. Tao, B. Narahari, and T. C. Zhao, "Heuristics for mapping parallel computations to heterogeneous parallel architectures," *in Proc. 1993 Workshop on Heterogeneous processing*, pp.36-41, 1993.

[13] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Computer*, Vol.37, pp.1384-1397, Nov. 1988.

[14] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE Computer*, Vol.15, pp.50-56, Jun. 1982.

[15] N. S. Bowen, C. N. Nikolaou, and A. Ghafoor, "On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems," *IEEE Computer*, Vol.41, pp.257-273, Mar. 1992.

[16] C. Leangsuksun and J. Potter, "Designs and experiments on heterogeneous mapping heuristics," *in Proc. 1994 Workshop on Heterogeneous processing*, pp.17-22, 1994.

[17] M. A. Iverson and G. J. Follen, "Parallel existing applications in a distributed heterogeneous environment," *in Proc. 1995 Workshop on Heterogeneous processing*, pp.93-100, 1995.

[18] B. Kruatrachue, and T. Lewis, "Grain size determination for parallel processing," *IEEE Software*, pp.23-32, Jan. 1988.

[19] T. Yang and A. Gerasoulis, "DSC: Scheduling tasks on an unbounded number of processors," *IEEE Trans. Parallel Distributed Systems*, Vol.5, pp.951-967, Sep. 1994.

[20] H. Kasahara and S. Narita, "An approach to supercomputing using multiprocessor scheduling algorithms," *in Proc. 1st International Conference Supercomputing Systems*, pp.16-20, Dec. 1985.

[21] K. Hwang and D. Degroot, *Parallel Processing for Supercomputers and Artifical Intelligence*, McGraw Hill Book Company, 1989.

[22] R. R. Muntz and E. G. Coffman, "Optimal preemptive scheduling on two-processor systems," *IEEE Trans. Computers*, Vol.C-18, pp.1014-1020, Nov. 1969.

[23] C. L. Chen, C. S. G. Lee, and E. S. H. Hou, "Efficient scheduling algorithms for robot inverse dynamic computation on a multiprocessor sys-

tem," *IEEE Trans. Systems, Man, Cybernetics*, Vol.18, pp.729-743, Sep. 1988.

[24] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, Massachusetts, 1994.

[25] Message Passing Interface Forum, *MPI:A Message-Passing Interface Standard, May*, 1994.

[26] S. Manoharan, "Augmenting work greedy assignment schems with task duplication," *ICPADS'97*, pp.772-777, 1997.

문 현 주

1995년 충북대학교 컴퓨터과학과 졸업 (이학사)

1997년 충북대학교 대학원 전자계산학과 졸업 (이학석사)

1997년~현재 충북대학교 대학원 전자계산학과 박사과정

관심분야 : 분산처리, 이기종 컴퓨팅

전 중 남

1981년 연세대학교 전자공학과 졸업(공학사)

1985년 연세대학교 대학원 전자공학과 졸업(공학석사)

1990년 연세대학교 대학원 전자공학과 졸업(공학박사)

1986년~1990년 연세대학교 산업기술 연구소 연구원

1990년~현재 충북대학교 컴퓨터과학과 조교수

관심분야 : 병렬컴퓨터 구조, 하드웨어 시뮬레이션

김 석 일

1975년 서울대학교 전기공학과 졸업(공학사)

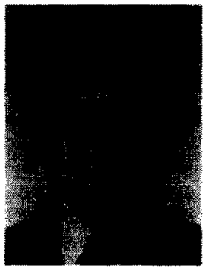
1975년~1990년 국방과학연구소 선임 연구원

1984년 연세대학교 대학원 전자계산학과 졸업(공학석사)

1989년 North Carolina State University 졸업(공학박사)

1990년~현재 충북대학교 컴퓨터과학과 부교수

관심분야 : 병렬처리 컴퓨터구조, 분산처리, 슈퍼컴퓨팅, 병렬처리 언어



황 인 재

1986년 충북대학교 컴퓨터공학과 졸업(공학사)

1991년 University of Florida, Computer & Information Sciences 졸업(공학석사)

1994년 University of Florida, Computer & Information Sciences 졸업(공학박사)

1986년~1987년 한국 전자통신연구소 연구원

1995년~현재 충북대학교 컴퓨터교육과 조교수

관심분야 : 병렬처리, 병렬컴퓨터 구조, 병렬 알고리즘