태스크 실행 시간을 최적화한 개선된 태스크 중복 스케줄 기법

(Modified TDS (Task Duplicated based Scheduling) Scheme Optimizing Task Execution Time)

장 세 이 † 김 성 천 ††
(Sei-Ie Jang) (Sungchun Kim)

요 약 최근 응용 프로그램들은 복잡한 데이타로 구성되어 있기 때문에 이를 효율적으로 처리할 수 있는 분산 메모리 기계(Distributed Memory Machine : DMM)의 필요성이 대두되었다. 특히 태스크 스케줄은 태스크 사이의 통신 시간을 최소화하여 응용 프로그램 전체의 실행 시간을 단축시키는 기법으로서, DMM의 성능을 향상시키는 매우 중요한 요소이다. 기존의 태스크 중복 스케줄(Task Duplicated based Scheduling : TDS) 기법은 두 개의 태스크 사이에 통신 시간이 많이 소요되는 것들을 하나의 클러스터(cluster)로 스케줄함으로써 통신 시간을 단축하여 실행 시간을 향상시키는 기법이다. 그러나 데이타를 전달하는 태스크와 이 태스크로 데이타를 전달받는 태스크 사이의 통신 시간을 최적화 하지 못하는 단점을 가진다. 따라서 본 논문에서는 이 두 태스크 사이의 최적화에 근접한 통신 시간을 갖는 개선된 중복 스케줄 (Modified Task Duplicated based Scheduling : MTDS) 기법을 제안하였다. 이 기법은 데이타를 전달한 태스크들을 클러스터링하기 위해 데이타를 전달받은 태스크에서 최적화 조건을 적용하여 검사한다. 그 결과 태스크 사이의 통신 시간을 단축하여 전체 태스크 실행 시간을 최소화하였다. 또한 시스템의 모델링을 통하여 MTDS 기법이 최상의 경우 TDS 기법보다 태스크 실행 시간을 70% 단축 시켰고 최악의 경우 TDS 기법과 동일한 실행 시간을 얻으므로 제안된 기법이 기존의 기법보다 우수함을 입증하였다.

Abstract Distributed Memory Machine(DMM) is necessary for the effective computation of the data which is complicated and very large. Task scheduling is a method that reduces the communication time among tasks to reduce the total execution time of application program and is very important for the improvement of DMM. Task Duplicated based Scheduling(TDS) method improves execution time by reducing communication time of tasks. It uses clustering method which schedules tasks of the large communication time on the same processor. But there is a problem that cannot optimize communication time between task sending data and task receiving data. Hence, this paper proposes a new method which solves the above problem in TDS. Modified Task Duplicated based Scheduling(MTDS) method which can approximately optimize the communication time between task sending data and task receiving data by checking the optimal condition, resulted in the minimization of task execution time by reducing the communication time among tasks. Also system modeling shows that task execution time of MTDS is about 70% faster than that of TDS in the best case and the same as the result of TDS in the worst case. It proves that MTDS method is better than TDS method.

1. 서 론

최근 응용 프로그램들은 복잡한 데이타로 구성되어 있기 때문에 이를 효율적으로 처리할 수 있는 분산 메모리 기계(Distributed Memory Machine :DMM)의 필요성이 대두되었다. 특히 태스크 스케줄은 태스크 사이의 통신 시간을 최소화하여 응용 프로그램 전체의 실행 시간을 단축시키는 기법으로서, DMM의 성능을 향상시

† 비 회 원 : 서강대학교 컴퓨터학과
bluesky@arqlab1.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
ksc@arqlab1.sogang.ac.kr
논문접수 : 1999년 2월 24일
심사완료 : 2000년 3월 8일

키는 매우 중요한 요소이다. 태스크 스케줄 기법은 태스크 분할 기법을 이용하여 만들어진 태스크들을 방향성을 갖으면서 사이클이 존재하지 않는 그래프(Directed Acyclic Graph :DAG)형태로 스케줄하는 것이다. DAG 그래프의 노드(node)와 에지(edge)는 태스크 그리고 데이타 상속성을 각각 나타낸다. 또한 각 노드는 태스크 실행시간, 각 에지는 테스크간 통신 시간을 나타낸다.

효율적인 태스크 스케줄 기법은 태스크를 스케줄할 때 태스크간 통신 시간을 최소화하여 전체 태스크 실행 시간을 최적화 하는 것을 목적으로 한다. 그러나 최적화 태스크 스케줄(optimal task scheduling) 문제는 NP-complete로 알려져 있다[1]. 따라서 특정한 조건을 만족할 경우 polynomial한 시간에 최적화된 태스크 실행 시간을 갖는 휴리스틱(heuristic) 기법들이 제안되고 있다. 이런 스케줄 기법들은 다음과 같다.

클러스터 스케줄(clustering scheduling) 기법은 태스크간 통신 시간이 큰 태스크를 선택하여 하나의 단위 즉, 클러스터로 묶어 스케줄 하는 기법이다. 동일한 클러스터에 있는 태스크들은 하나의 프로세서로 스케줄되어 실행됨으로써 프로세서 사이에서 발생하는 통신 시간이 제거된다. 그러나 이 기법은 태스크간 통신 시간이 큰 것을 제거하지만 최적화된 태스크 실행 시간을 보장하지 못한다. 태스크 중복 스케줄(Task Duplication based Scheduling : TDS)은 태스크가 여러 개의 자식 태스크를 가지고 있을 때 이런 태스크를 중복하여 서로 다른 클러스터에 스케줄하는 기법이다. 따라서 여러 개의 자식 노드를 가지고 있는 태스크가 많은 그래프를 효과적으로 스케줄 할 수 있다. 그러나 여러 개의 부모 태스크를 가지고 있는 태스크가 많은 그래프에서는 비효과적이다.

따라서 TDS 기법의 문제를 해결하기 위해 개선된 태스크 중복 스케줄 (Modified Task Duplicaed based Scheduling : MTDS) 기법을 제안한다. 이 기법은 부모 태스크를 클러스터링하여 부모 태스크 사이의 통신 시간을 최적화 한다. 부모 태스크를 갖는 태스크(조인 태스크: join task)에서 최적화 조건을 고려한다. 최적화 조건이란 태스크 실행 시간 비율이 태스크 통신 시간 비율 보다 큰 것을 의미한다. 따라서 각각의 부모 태스크를 최대한 병렬적으로 수행하여 실행 시간을 단축하는 것이 전체 태스크 실행 시간을 최소화할 수 있는 방법이다.

그러므로 MTDS 기법은 부모 태스크를 서로 다른 클러스터로 스케줄하여 각각 다른 프로세서에서 병렬적으로 수행하게 한다. 또한 최적화 조건을 만족하지 않음은 실행 시간 비율이 통신 시간 비율 보다 작다는 것을 의미하므로 이 경우에는 부모 태스크와 자식 태스크간 통신 시간을 단축하는 것이 전체 태스크 실행 시간을 최소화할 수 있는 방법이다. 따라서 부모 태스크를 동일한 클러스터에 스케줄하여 부모 태스크와 자식 태스크간 통신 시간을 단축한다. MTDS 기법은 모든 조인 태스크에서 최적화 조건을 검사하여 그 경우에 맞는 효과적인 스케줄을 함으로써 최적화된 태스크 실행 시간을 갖는 스케줄 결과를 나타낸다. TDS 기법과 MTDS 기법을 구현하여 시뮬레이션 한 결과 MTDS 기법이 TDS 기법보다 최대 70%정도 전체 태스크 실행 시간을 단축시킴을 확인하였다.

2. 중복 태스크 스케줄 기법

이번 장에서는 TDS 기법에서 사용되는 용어와 가정에 대해 설명한다. 이 용어와 가정은 MTDS 기법에서도 동일하게 적용되며 기존의 태스크 스케줄 기법 연구에서도 공통적으로 사용되는 것이다.

2.1 태스크 스케줄 모델

태스크 실행 시간과 통신시간: 태스크 실행 시간은 하나의 태스크가 처리될 동안의 프로세스 점유 시간을 의미하며 단위는 프로세서 시간 단위와 같다. 또한 통신 시간은 태스크 사이에서 전송되는 데이타의 크기 즉 메시지의 길이에 비례하여 이 메세지를 프로세서에서 처리하는 시간으로 가정한다. 따라서 통신 시간은 고정된 길이의 메세지에 대해 동일하고 메세지의 크기에 비례한다. 그 밖의 요소는 무시한다.

태스크 중복 스케줄의 가정: 태스크를 프로세서로 스케줄할 때 프로세서의 개수는 무제한으로 가정한다. 또한 모든 태스크는 각 프로세서의 메모리에 로드 되도록 메모리의 크기는 제한이 없다고 한다. 이런 가정을 함으로써 하나의 태스크는 동일한 초기 데이타를 가지고 서로 다른 프로세서에서 동시에 실행이 가능하다. 따라서 시스템에 오류가 발생하지 않는다면 하나의 태스크는 서로 다른 프로세서에서 동시에 실행되어도 결과는 서로 간에 일관성을 유지한다.

2.2 태스크 중복 스케줄 기법(TDS)

2.2.1 TDS 기법

이 기법은 클러스터 스케줄 기법에 바탕을 둔다. 즉 태스크 사이에 존재하는 통신 시간이 큰 두 개의 태스크를 선택하여 하나의 단위 즉, 클러스터로 묶어 스케줄하는 기법이다. 그러나 하나의 태스크가 다른 클러스터에 있는 여러 개의 태스크로 데이타를 보낼 때 통신 시간이 큰 경우 문제가 발생한다. 이를 해결하기 위해

TDS 기법은 조인 태스크가 있는 각각의 클러스터에 부모 태스크를 중복하여 스케줄 함으로써 태스크간 통신 시간을 최소화한다. 다음에 정의된 파라미터를 이용하여 TDS 기법은 태스크 스케줄을 한다. 각 파라미터는 DAG의 태스크 i를 기준으로 표현된다[2,3].

Def 1. 태스크 i에 대한 부모 태스크:

- $pred(i)=\{j \mid e_{i,j}\in E\}$

Def 2. 태스크 i에 대한 자식 태스크:

- $succ(i)=\{j \mid e_{i,j}\in E\}$

Def 3. 최초 태스크 실행 시작 시간:

태스크 i가 실행이 시작되는 최초의 시간(earliest start time)

- $est(i)=0$, if $pred(i)=\phi$
- $est(i)=\min_{j\in pred(i)} \max_{k\in pred(i),k\neq j}(ect(j), ect(k)+c_{k,j})$, if $pred(i)\neq\phi$

Def 4. 최초 태스크 실행 완료 시간:

태스크 i가 실행 완료되는 최초의 시간(earliest completion time)

- $ect(i)=est(i)+\tau(i)$

Def 5. 우선 순위 부모 태스크:

태스크 i의 부모 태스크 중 실행 시간과 통신 시간의 합이 가장 큰 태스크(favorite predecessor)

- $fpred(i)=\{j \mid (ect(j)+c_{j,i})\geq(ect(k)+c_{k,i})\}$, $\forall j\in pred(i);\ k\in pred(i), k\neq j$

Def 6. 최종 태스크 실행 완료 시간:

태스크 i가 실행 완료되는 마지막 시간(last completion time)

- $lact(i)=ect(i)$, if $succ(i)=\phi$
- $lact(i)=\min(\min_{j\in succ(i), i\neq fpred(j)}(last(j)-c_{i,j}), \min_{j\in succ(i), i=fpred(j)}(last(j)))$

Def 7. 최종 태스크 실행 시작 시간:

태스크 i가 실행 시작 될 수 있는 마지막 시작 시간 (last start time)

- $last(i)=lact(i)-\tau(i)$

Def 8. 끝 태스크부터 태스크 i까지 실행 시간의 합:

- $level(i)=\tau(i)$, if $succ(i)=\phi$
- $level(i)=\max_{k\in succ(i)}(level(k)+\tau(i))$, if $succ(i)\neq\phi$

위의 파라미터들은 다음 방식으로 계산되어진다. Def 1-5는 시작 태스크에서 끝 태스크로 탐색되어 각 태스크에 대한 파라미터가 계산되는 톱-다운(top-down)방식으로 진행된다. 반면 Def 6-8은 끝 태스크에서 시작 태스크로 탐색되어 계산되는 바텀-업(bottom-up) 방식으로 진행된다.

다음은 TDS 기법의 pseudo code를 나타낸 것이다. 스텝 1-3을 수행하면 모든 태스크가 클러스터링되고 각 태스크는 알맞은 프로세서로 스케줄되는 것이다.

TDS 기법의 pseudo code

```
입력:
DAG(V,E, τ ,c)
pred(i): 태스크 i의 부모 태스크 집합.
succ(i): 태스크 i의 자식 태스크 집합.

출력:태스크스케줄
시작
스텝 1.모든 태스크에 대한 est(i), ect(i), 그리고 fpred(i)를 계산함.
스텝 2.모든 태스크에 대한 last(i), lact(i), 그리고 level(i)를 계산함.
스텝 3.TDS기법을 사용하여 모든 태스크를 프로세서에 스케줄함
끝
```

3. 개선된 중복 태스크 스케줄 기법

DAG는 포크(fork) 태스크와 조인(join) 태스크로 구성된다. 포크 태스크는 하나의 태스크에서 여러 개의 자식 태스크로 데이타를 전달한다. 그리고 조인 태스크는 두개 이상의 부모 태스크로부터 데이타를 받는다. 따라서 포크 태스크와 조인 태스크를 스케줄하는 방법은 차이를 보일 수밖에 없다.

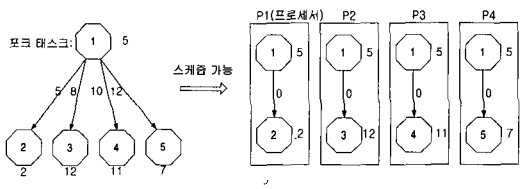


그림 1 포크 태스크의 스케줄

포크 태스크를 스케줄하는 방법은 다음과 같다. 각 태스크는 부모 태스크에서 데이타를 받는 즉시 실행될 수 있으므로 [그림 1]과 같이 포크 태스크를 중복하여 각각의 자식 태스크에 클러스터링하고 각각의 프로세서로 스케줄함으로써 포크 태스크와 자식 태스크간 통신 시간을 최소화하였다.

그러나 조인 태스크는 모든 부모 태스크에서 데이타가 도착할 때 실행될 수 있다. 따라서 조인 테스크는 [그림 2]와 같이 스케줄 될 수 없다. 즉, 오직 하나의 부모 태스크만이 조인 태스크와 동일한 프로세서로 할

당된다.

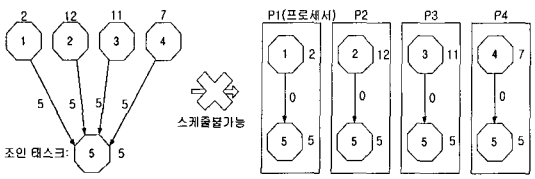


그림 2 조인 태스크의 스케줄 불가능

그러므로 나머지 부모 태스크와 조인 태스크 사이에 통신 시간을 최소화 할 수 없다. 이 문제를 해결하기 위해 각 조인 태스크에서 최적화 조건을 검사한다.

3.1 최적화 조건

조인 태스크와 부모 태스크 사이에 통신 시간을 최적화하기 위해서는 각 조인 태스크에서 최적화 조건을 검사하여야 한다. 그 최적화 조건의 정의는 다음과 같다.

최적화 조건의 정의: 태스크 i의 부모 태스크 중에 $\{(ect(j)+c_{j,i}) \mid j \in pred(i)\}$의 값이 첫째, 둘째로 큰 태스크를 각각 m과 n이라 하자. 이때 다음 조건을 만족하면 태스크 i는 최적화 조건을 만족한다고 한다.

- $\tau(m) \geq c_{n,i}$ if $est(m) \geq est(n)$ *or*,
- $\tau(m) \geq (c_{n,i} + est(n) - est(m))$ if $est(m) < est(n)$.

태스크 i가 최적화 조건을 만족할 때 태스크 m과 n은 태스크 실행 시간의 비율이 i와의 통신 시간보다 크다. 따라서 m과 n을 각각 다른 클러스터에 스케줄하여 병렬적으로 실행하는 것이 i를 더 빨리 실행시킬 수 있다. 이것을 증명하면 다음과 같다.

정리. 조인 태스크가 최적화 조건을 만족하면, 부모 태스크들이 서로 다른 클러스터로 스케줄 되는 것은 조인 태스크의 실행 시작 시간을 단축시킨다 [4].

Proof: [그림 3]은 조인 태스크의 예이다. 최적화 조건에 따라 m과 n은 $\{ect(j)+c_{j,i}, j \in pred(i)\}$의 값이 각각 첫 번째 그리고 두 번째로 큰 태스크이다. 그리고 태스크 m은 프로세서 m에 태스크 n은 프로세서 n에 스케줄 된다고 가정한다. 태스크 m의 $ect(m)+c_{m,i}$ 값이 가장 크기 때문에 태스크 i와 태스크 m은 동일한 프로세서인 프로세서 m에 할당되고 $est(i) = \max\{ect(m), ect(n) + c_{n,i}\}$이다. 또한 m과 n을 제외한 다른 부모 태스크들 역시 임의의 실행 시간과 통신 시간을 갖지만 $ect(m)$ or $ect(n)+c_{n,i}$보다는 작은 값을 갖는다. 따라서 이들 부모 태스크들은 $ect(m)$ or $ect(n)+c_{n,i}$ 안에 모두 실행되므로 태스크 i의 최초 실행 시작 시간에 영향을 주지 않는다. 그러므로 여기서는 태스크 i와 부모 태스크 m과 n만을 고려한다

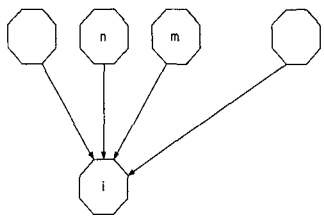


그림 3 조인 태스크의 예

부모 태스크인 m과 n의 관계에 대한 경우의 수는 다음과 같이 두 가지이다.

경우 1 : $est(m) \geq est(n)$. 최적화 조건을 만족하므로 $\tau(m) \geq c_{n,i}$가 성립된다. 이것은 두 가지 경우로 다시 나눌 수 있다.

경우 1-1) $ect(m) \geq ect(n) + c_{n,i}$, *i.e.*, $est(i) = ect(m)$.

만약 태스크 m, n 그리고 i 모두 동일한 프로세서에 할당된다면, $est(i) = \max(est(m), (est(n) + \tau(n))) + \tau(m) = \max(ect(m), ect(n) + \tau(m))$,즉 est(i) = ect(m)이다. 그러므로 $est(i)$는 $ect(m)$이하로 단축될 수 없다.

경우 1-2) $ect(m) < ect(n) + c_{n,i}$, *i.e.*, $est(i) = ect(n) + c_{n,i}$.

만약 태스크 m, n 그리고 i 모두 동일한 프로세서에 할당된다면 태스크 i의 $est(i)$는 다음과 같다. $est(i) = est(n) + \tau(n) + \tau(m) = est(i) = ect(n) + \tau(m)$이다. 그러므로 $\tau(m) \geq c_{n,i}$ 조건에 의해 est(i)는 $ect(n) + \tau(m) \geq ect(n) + c_{n,i}$을 만족한다. 따라서 $est(i)$는 단축되지 않는다.

경우 2 : $est(m) < est(n)$. 최적화 조건을 만족하므로 $\tau(m) \geq c_{n,i} + est(n) - est(m)$가 성립된다. 이것은 두 가지 경우로 다시 나눌 수 있다.

경우 2-1) $ect(m) \geq ect(n) + c_{n,i}$, *i.e.*, $est(i) = ect(m)$.

만약 태스크 m, n 그리고 i 모두 동일한 프로세서에 할당된다면, $est(i) = est(m) + \tau(m) + \tau(n) = ect(m) + \tau(n)$ 이다. 그러므로 $est(i)$는 $ect(m)$이하로 단축되지 않는다.

경우 2-2) $ect(m) < ect(n) + c_{n,i}$, *i.e.*, $est(i) = ect(n) + c_{n,i}$.

만약 태스크 m, n 그리고 i 모두 동일한 프로세서에 할당된다면 태스크 i의 $est(i)$는 다음과 같다. $est(i) = est(m) + \tau(m) + \tau(n)$. 만약 다음 두 조건 $est(m) + \tau(m) < est(n)$

$+ c_{n,i}$ 또는 $r(m) < (est(n) - est(m)) + c_{n,i}$이 만족된다면 $est(i)$는 줄어들 수 있다. 그러나 $r(m) \geq (est(n) - est(m)) + c_{n,i}$라고 가정되었으므로 $est(i)$는 단축되지 않는다[4]. 증명끝

3.2 TDS 기법의 문제점

모든 조인 태스크가 최적화 조건을 만족하면 TDS 기법은 최적화된 태스크 실행 시간을 나타낸다. 그 이유는 TDS 기법은 조인 태스크의 부모 태스크를 무조건 다른 클러스터로 스케줄하기 때문이다. 그러나 통신 시간 비율이 실행 시간 비율보다 클 경우에는 조인 태스크에서 최적화 조건이 만족되지 않는다. 이것은 TDS 기법에서 문제를 발생시킨다. 즉 최적화 조건을 만족하지 않는 조인 태스크에 대해 TDS 기법은 무조건적인 클러스터링 작업을 하여 조인 태스크의 실행 시작 시간을 단축시키지 못하여 비효율적인 태스크 실행 시간을 나타낸다.

[그림 4]은 태스크 6에서 최적화 조건을 만족하지 않는 예이다.

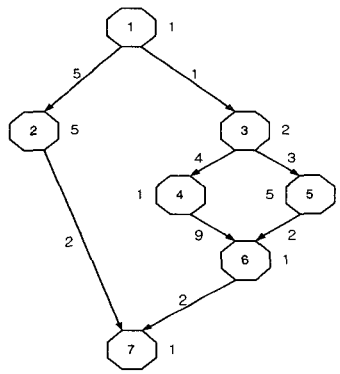


그림 4 최적화 조건을 만족하지 않는 조인 태스크의 예

표 1 TDS 기법에서 사용될 태스크의 파라미터의 값

태스크	est(i)	ect(i)	fpred(i)	last(i)	lact(i)	level(i)
1	0.00	1.00	-	0.00	1.00	10.00
2	1.00	6.00	1	4.00	9.00	6.00
3	1.00	3.00	1	1.00	3.00	9.00
4	3.00	4.00	3	9.00	10.00	3.00
5	3.00	8.00	3	3.00	8.00	7.00
6	10.00	11.00	4	10.00	11.00	2.00
7	11.00	12.00	6	11.00	12.00	1.00

[표 1]은 TDS 기법의 스텝 1과 2를 수행하여 각 태스크에 대한 파라미터를 계산한 결과이다. 조인 태스크 6의 최적화 조건을 검사하면, $est(4) \geq est(5)$일 때 $r(4) < c_{5,6}$인 결과를 나타내어 태스크 6은 최적화 조건을 만족하지 않는다.

다음 [그림 5]는 TDS 기법을 스텝 3에 적용하여 스케줄한 결과이다. 이 때 3개의 클러스터가 만들어졌다. 이 3개의 클러스터를 병렬적으로 실행시키면 [표 2]에서 확인할 수 있듯이 전체 태스크 실행 시간은 11이 된다.

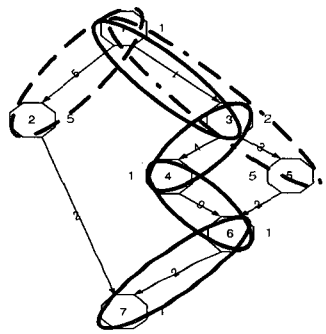
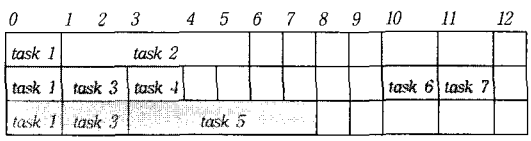


그림 5 TDS 기법의 적용 결과

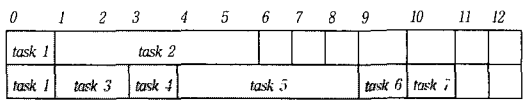
TDS 기법은 최적화 조건을 만족하지 않는 조인 태스크 6의 부모 태스크 4와 5를 서로 다른 태스크로 스케줄 하였다. 그러나 태스크 4와 5가 하나의 클러스터로 스케줄 되면 조인 태스크 6의 실행 시작 시간을 단축시킬 수 있다. 그것을 [표 3]을 통해 확인할 수 있다.

표 2 [그림 4]에 TDS 기법을 적용한 스케줄 결과



[표 3]에서는 태스크 4와 5를 동일한 클러스터에 스케줄하여 태스크 6의 실행 시작 시간을 10에서 9로 단축하여 전체 태스크 실행 시간을 10으로 단축하였다.

표 3 [그림 4]의 태스크 4와 5를 하나의 클러스터에 스케줄한 결과



TDS 기법은 조인 태스크에서 최적화 조건을 고려하지 않고 무조건 부모 태스크를 다른 클러스터로 스케줄하여 비효율적인 태스크 스케줄을 하는 문제점이 있다. 이 문제를 해결하기 위해 모든 조인 태스크에서 최적화 조건을 검사하여 효율적인 스케줄을 하는 기법에 대해 소개한다.

3.3 개선된 태스크 중복 스케줄 기법(MTDS)

TDS 기법의 문제점을 해결하기 위해 MTDS 기법을 제안한다. 이 기법은 조인 태스크에서 최적화 조건을 검사하여 최적화 조건을 만족하면 기존의 TDS 기법을 적용하여 부모 태스크들을 서로 다른 클러스터로 스케줄한다. 또한 최적화 조건을 만족하지 않으면 최초 태스크 실행 완료 시간과 통신 시간의 합이 큰 두 개의 부모 태스크를 하나의 클러스터로 스케줄한다.

다음은 MTDS 기법의 pseudo code를 나타낸 것이다. 스텝 1-3을 수행함으로써 조인 태스크와 부모 태스크를 최적화된 결과로 스케줄한 결과를 얻는다.

MTDS 기법의 pseudo code

```
입력:
DAG(V,E, τ,c)
pred(i): 태스크 i의 부모 태스크 집합
succ(i): 태스크 i의 자식 태스크 집합

출력:태스크스케줄
시작
스텝 1.모든 태스크에 대한 est(i), ect(i), fpred(i) 그리고 spred(i)를 계산함
스텝 2.모든 태스크에 대한 last(i), lact(i), 그리고 level(i)를 계산함
스텝 3.MTDS기법을 사용하여 모든 태스크를 프로세서에 스케줄함
끝
```

MTDS 기법에는 조인 태스크에서 최적화 조건을 검사하는 기능이 첨가되었다. 스텝 1과 2는 TDS 기법과 동일하지만 스텝 1에서 spred라는 새로운 파라미터를 추가한다.

Def 9. 두 번째 우선 순위 부모 태스크 :
조인 태스크 i의 부모 태스크 중 실행 시간과 통신 시간의 합이 두 번 째로 큰 태스크(second favorite predecessor)

- $spred(i)=j \mid (ect(j)+c_{j,i}) \geq (ect(k)+c_{k,i})$,
  $\forall j \in pred(i);\ k \in pre(i), k \neq j$ and $k \neq fpred(i)$, if $\mid pred(i) \mid \geq 2$
- $spred(i)=fpred(i)$, *otherwise*

다음은 MTDS 기법의 스텝 3을 구체화 한 것이다. 스텝 3는 TDS 기법과 동일하다. 그러나 밑줄 그어진 부분이 MTDS 기법에 첨가되었다. 즉 MTDS 기법은 조인 태스크에서 최적화 조건을 검사한다. 이 기법은 선택된 태스크가 조인 태스크인지를 조사한다. 즉, 선택된 태스크의 우선 순위 부모 태스크(fpred)와 두 번째 우선 순위 부모 태스크(spred)가 동일하면 조인 태스크로 판명한다. 태스크 x가 조인 태스크로 판명되면 최적화 조건이 검사된다.

MTDS 기법의 스텝 3

```
출력:태스크 클러스터
시작
x = 배열 큐의 첫번째 태스크
i = 0;
x를 프로세서 P_i에 스케줄
while (모든 태스크가 프로세서에 할당되지 않음){
  y = fpred(x)
  if( y가 이미 다른 프로세서에 스케줄됨) {
     if( (last(x) − lact(y)) ≥ c_{x,y}) /*y가 x에 대해 critical path가
       아님*/
       y = 어떤 프로세서에도 스케줄 되지 않은 x의 부모
           태스크
     else
       z는 x의 다른 부모 태스크이고, y≠ z
       if( (ect(y)+ c_{y,x})=(ect(z)+ c_{z,x}) 이고 z가 어떤 프로세
         서에도 스케줄되지 않았을 때)  y = z
  }
if (x는 조인 태스크이다)  이면          {              /*
  fpred(x) == spred(x) */
   m=              fpred(x);          n              =spred(x);
/*MTDS 기법에 첨가된 코드*/
  if(x가 최적화 조건을 만족 안함 ) 이면  /*MTDS 기법에 첨
    가된 코드*/
    m과 n을 프로세서 P_i로 스케줄      /*MTDS 기법에 첨
      가된 코드*/
  }
y를 프로세서 P_i에 스케줄
x = y
if x가 시작 태스크이면 {
     x를 프로세서 P_i에 스케줄
     x = 배열 큐에서 아직 프로세서로 스케줄 되지 않는
         level 값이 작은 태스크
     i++
     x를 프로세서 P_i에 스케줄
 }
}
끝
```

이때 우선 순위 부모 태스크와 두번째 우선 순위 부모 태스크를 각각 m과 n으로 지정한다. 만약 x에서 최적화 조건을 만족하지 않을 경우 m과 n을 프로세서 P_i에 동시에 스케줄 한다. x에서 최적화 조건을 만족할 경우에는 TDS 기법과 동일하게 m과 n을 다른 클러스터로 스케줄 한다.

3.3.1 MTDS 기법의 복잡도

MTDS 기법이 최적화 태스크 스케줄 기법에 근접한 결과를 보이는 것은 매우 중요하다. 그러나 이런 결과를 얻기 위한 방법의 복잡도가 크다면 이것은 의미가 없다. 따라서 3.3.1에서는 MTDS 기법의 복잡도를 계산한다. MTDS 기법의 스텝 1과 2는 DAG의 모든 태스크를 각각 한번씩 방문하여 파라미터들을 계산한다. 또한 각 태스크에서 입력 에지와 출력 에지가 검사되므로 최악

의 경우 DAG의 모든 에지가 검사된다. 따라서 스텝 1과 2의 최악의 복잡도는 $O(|E|)$이다. (|E|는 에지의 개수). 스텝 3에서는 태스크 방문 방법이 깊이 우선 검색(depth first search)방식으로 진행하기 때문에 스텝 3의 복잡도는 일반적인 검색 알고리즘과 같은 $O(|V|+|E|)$이다[7]. (|V|는 태스크의 개수) 또한 스텝 3에서는 태스크를 level 값이 작은 순서에서 큰 순서로 배열 큐에 정렬해야 한다. 이것은 $O(|V|\log|V|)$에 해결 가능하다. 그러므로 MTDS 기법의 전체 복잡도는 $O(|V|+|E|+|V|\log|V|)$가 된다. 노드가 밀집된 그래프에서는 에지의 개수가 $O(|V|^2)$에 비례한다. 따라서 최악의 경우에 해당되는 MTDS 기법의 복잡도는 $O(|V|^2)$가 된다.

3.3.2 MTDS 기법의 적용 예

[그림 4]를 모델로하여 MTDS 기법을 적용한다.

스텝 1) pred(1) = ϕ이기 때문에 시작 태스크의 est는 0이다. 태스크 1은 time 1만에 수행된다. 다른 태스크들의 est와 ect는 Def 3,4를 이용하여 구할 수 있다. 예를 들어 조인 태스크 6의 최초 실행 시작 시간인 est(6)을 계산하자. 태스크 6의 부모 태스크는 태스크 4와 5이다. 이때 Def 3에 의해 태스크 6의 est는 다음과 같다. $est(6)=\min\{\max\{(ect(4)+c_{4,6}), ect(5)\}\ \max\{(ect(5)+c_{5,6}), ect(4)\}\}=10$. 그리고 fpred와 spred는 Def 5,9를 이용한다. 모든 태스크들의 파라미터 값의 결과는 [표 4]와 같다.

스텝 2) 끝 태스크로부터 시작하여 lact와 last 그리고 level은 각각 Def 6-8로 계산된다. 예를 들어 태스크 3의 최종 실행 완료 시간인 lact(3)를 계산하자. Def 6에 의해 last(3)는 다음과 같다.

$$lact(3)=\min_{fpred(4),fpred(5)=3}(last(4), last(5))=\min(9,3)=3.$$

모든 태스크들의 파라미터 값의 결과는 [표 5]와 같다.

표 4 MTDS 기법에서 사용된 파라미터 값의 결과

태스크	est(i)	ect(i)	fpred(i)	spred(i)	last(i)	lact(i)	level(i)
1	0.00	1.00	-	-	0.00	1.00	10.00
2	1.00	6.00	1	1	4.00	9.00	6.00
3	1.00	3.00	1	1	1.00	3.00	9.00
4	3.00	4.00	3	3	9.00	10.00	3.00
5	3.00	8.00	3	3	3.00	8.00	7.00
6	10.00	11.00	4	5	10.00	11.00	2.00
7	11.00	12.00	6	2	11.00	12.00	1.00

스텝 3) 배열 큐는 다음과 같이 구성된다. 배열 큐 = {7,6,4,2,5,3,1}. 배열 큐의 첫 번째 태스크인 7을 첫 번째 클러스터로 스케줄한다. 그 다음 7의 우선 순위 부모 태스크인 6을 그 클러스터로 스케줄한다. 그러나 6은 조인 태스크이다. MTDS 기법은 최적화 조건을 검사하는데 태스크 6은 $est(4)\geq est(5)$일 때 $\tau(4)<c_{5,6}$를 나타내어 최적화 조건을 만족하지 않는다. 따라서 부모 태스크인 4와 5가 동일한 클러스터로 스케줄된다. 이러한 클러스터링 작업으로 태스크가 7-6-4-5-3-1인 첫번째 클러스터와 태스크가 2-1인 두번째 클러스터가 만들어진다.

표 5 [그림 4]에 MTDS 기법을 적용한 스케줄 결과

0	1	2	3	4	5	6	7	8	9	10	11	12
task 1	task 2											
task 1	task 3		task 4		task 5				task 6	task 7		

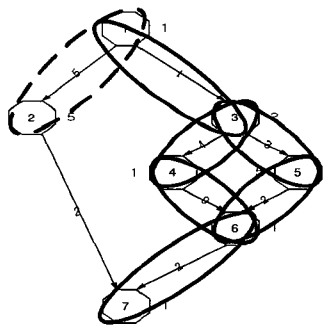


그림 6 MTDS 기법의 적용 결과

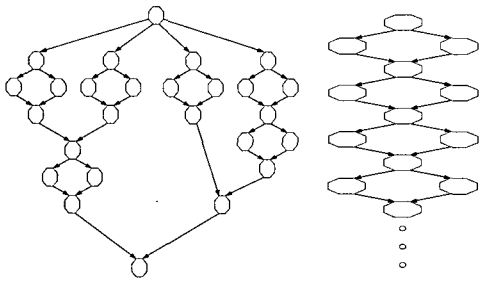
[표 5]는 태스크의 실행 시간을 나타낸 것이다. 두개의 클러스터를 생성하여 전체 태스크 실행 시간이 10인 결과를 보여준다. 이때 최적화된 태스크 실행 시간인 시작 태스크 1의 level이 10이다. 이것의 의미는 MTDS 기법이 이 모델에서 최적화된 태스크 실행 시간으로 태스크를 스케줄함을 알 수 있는 것이다. [그림 6]은 생성된 두개의 클러스터를 나타내고 있다.

3.3.3 MTDS 기법의 최상 조건

DAG의 구조는 여러 가지의 형태가 있다. 일반적인 DAG의 구조는 [그림 7.(a)]처럼 불규칙적인 조인 태스크의 형태를 갖는다. 즉, 서로 다른 클러스터에 있는 태스크로부터 데이타를 받는 것이다. 그러나 [그림 7.(b)]는 조인 태스크로 모든 데이타가 입력되고 출력되는 구조를 나타낸다. 이런 구조를 다이아몬드 구조라고 하자.

다이아몬드 구조는 조인 태스크의 비율이 높다는 특징을 나타낸다. MTDS 기법은 조인 태스크의 실행 시작 시간을 단축하여 전체 태스크 실행 시간을 최소화시

킴으로 조인 태스크의 비율이 높은 구조에서 효과적인 성능을 나타내는 것이다.



(a) 일반구조 DAG (b) 다이아몬드 구조 DAG

그림 7 일반 구조 DAG 모델과 다이아몬드 구조 DAG 모델

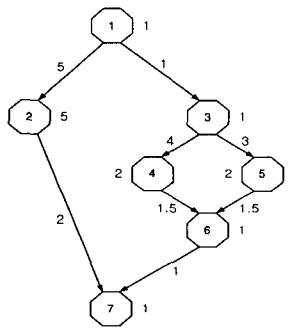


그림 8 기본 DAG 모델

표 6 기존 스케줄 기법과 MTDS 기법의 비교

스케줄 기법	*전체 실행 시간*	*복잡도*
Optimal Algorithm	8.5	NP-complete
Linear Clustering	11.5	$O(\|V\|(\|E\|+\|V\|))$
MCP Algorithm	10.5	$O(\|V\|^2 \log \|V\|)$
Internalization Prepass	10.0	$O(\|E\|(\|V\|+\|E\|))$
Dominant Sequence Clustering	9.0	$O((\|V\|+\|E\|)\log \|V\|)$
Threshold Scheduling Algorithm	10.0	$O(\|V\|^2)$
TDS Algorithm	8.5	$O(\|V\|^2)$
MTDS Algorithm	8.5	$O(\|V\|^2)$

3.4 다른 태스크 스케줄 기법과 비교

기존의 스케줄 기법인 Linear Clustering[1], TDS[4], DSC[5], Internalization Prepass[6], MCP Clustering[7]들은 모두 [그림 7]을 기반으로 연구되었다. [표 6]은 [그림 8]을 모델로 하여 MTDS 기법과 기존 스케줄 기법을 비교한 결과이다.

4. 성능 평가

MTDS 기법의 성능을 평가하기 위해 [그림 7]의 구조인 DAG를 모델로하여 TDS 기법과 MTDS 기법을 구현하였다. 두 기법을 비교하기 위해 태스크 실행 시간과 통신 시간의 비율인 그래인 크기를 변화시켰다. 그 이유는 그래인 크기가 크면 최적화 조건을 만족하는 조인 태스크의 개수는 증가하고 그래인 크기가 작으면 최적화 조건을 만족하는 조인 태스크의 개수는 감소한다. 그래인 조인 태스크에 변화를 주기 위해서 그래인 크기가 큰 경우에는 태스크 실행 시간 범위를 0-30 , 통신 시간 범위를 0-10로 제한하였고 그래인 크기가 작은 경우에는 실행 시간을 0-10 , 통신 시간을 0-30까지로 제한한다.

시뮬레이션에서 TDS 기법과 MTDS 기법은 C 언어로 구현하였다. 태스크의 구조는 [그림 7.(a)]인 일반 구조 DAG 와 [그림 7.(b)]의 다이아몬드 구조 DAG를 사용하였다. 각 DAG의 태스크 실행 시간과 통신 지연 시간은 난수 발생기인 random 함수를 사용하여 지정하였고 난수 발생기의 seed 값으로 현재 시간 값을 리턴하는 time함수를 사용하였다. 위 시뮬레이션은 태스크의 실행 시간과 통신 시간을 변화시켜 1000번씩 독립적으로 실행하였다.

표 7 TDS 기법과 MTDS 기법의 성능 평가

DAG 모델 / 그래인 크기	다이아몬드 구조			일반 구조		
	TDS	MTDS	성능	TDS	MTDS	성능
태스크실행시간 범위:10 태스크통신시간 범위:30 (그래인 크기가 크다)	473.3455	388.88	69.95%	117.5138	105.907	9.87%
태스크실행시간 범위:30 태스크통신시간 범위:10 (그래인 크기가 작다)	508.7608	500.3873	1.64%	139.225	137.0231	1.58%

5. 결 론

분산 메모리 기계는 복잡한 데이타로 구성되어 있는 응용프로그램들을 효율적으로 처리하기 위해 필요한 시스템이다. 특히 태스크 스케줄은 태스크 사이의 통신 시간을 최소화하여 응용 프로그램 전체의 실행 시간을 단축시키는 기법으로서, DMM의 성능을 향상시키는 매우

중요한 요소이다.

본 논문에서는 TDS 기법의 문제점을 해결하기 위해 MTDS 기법을 제안했다. 이 기법은 조인 태스크에서 최적화 조건을 검사하여 부모 태스크를 클러스터링 하는 것이다. 그 결과로 조인 태스크의 실행 시작 시간을 단축함으로써 전체 태스크 실행 시간을 최적화하였다.

MTDS 기법의 성능을 평가하기 위해 TDS 기법과 MTDS 기법을 모델링하였다. 즉 DAG 구조와 그래인 크기를 변수로하여 두 기법간 전체 태스크 실행 시간을 비교하였다. 이때 DAG가 다이아몬드 구조이면서 그래인 크기가 작은 경우, MTDS 기법이 TDS 기법보다 전체 태스크 실행 시간을 70%정도 향상 시켰다. 일반 구조이면서 그래인 크기가 클 때 약 1.5%만을 향상 시켰다. 또한 이 기법은 TDS 기법의 복잡도 $O(|V^2|)$를 유지하면서 전체 태스크 실행 시간을 향상함으로써 MTDS 기법이 TDS 기법보다 우수함을 실험을 통하여 입증하였다.

본 논문에서 제안된 MTDS 기법은 부모 태스크들이 하나의 조인 태스크에만 연결된 구조에 대해 개선된 스케줄 방법을 적용하였다. 그러나 부모 태스크들이 여러 개의 조인 태스크와 연결된 경우에 대해 개선된 스케줄 방법을 적용하는 연구가 필요하겠다.

참 고 문 헌

[1] S.J. Kim and J.C. Browne, "A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures," Proc. Int'l Conf. Parallel proces-sing, vol.3, pp.1-8, 1988

[2] S.S. Pande, D.P. Agrawal, and J. Mauney, "A New Threshold Scheduling Strategy for Sisal Programs on Distributed Memory Systems," J. Parallel and Distributed Computing, vol.21, no.2, pp.223-236, 1994

[3] T.L. Adam, K.M. Chandy, and J.R. Dickson, "A Comparison of List Schedules for Parallel Processing Systems," ACM Comm., vol.17, no.12, pp.685-690, 1974

[4] S. Darbha, and D.P. Agrawal, "Optimal Scheduling Algorithm for Distributed Memory Machines," IEEE Trans. Parallel and Distributed Systems, vol.9, no.1, pp.87-95, 1998

[5] A. Gerasoulis and T. Yang, "A Comparision of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors," J. Parallel and Distrib-uted Computing, vol.16, pp.276-291, 1992

[6] V. Sarkar, Partitioning and Scheduling Parallel programs for Execution on multiprocessors, MIT Press. Cambridge, MA, 1989

[7] M.Y. Wu and D. Gajski, "A Programming Aid for Hypercube Architectures," J. Supercomputing, vol.2, pp.349-372, 1988

장 세 이
1997년 서강대학교 공과대학 전자계산학과(학사). 1999년 서강대학교 전자계산학과 대학원(석사). 관심분야는 병렬 컴퓨터 구조, 태스크 스케줄링, 워크스테이션 클러스터링.

김 성 천
정보과학회논문지 : 시스템 및 이론
제 27 권 제 2 호 참조