Journal of the Korean Institute
of Industrial Engineers
Vol.24, N0.4, December, 1998

다중프로세서를 갖는 유방향무환그래프 모델의 스케쥴링을 위한 유전알고리즘을 이용한 선형 클러스터링 해법

A Linear Clustering Method for the Scheduling of the Directed Acyclic Graph Model with Multiprocessors Using Genetic Algorithm

성기석* · 박지혁*

Kiseok Sung* · Jeehyuk Park*

Abstract

The scheduling of parallel computing systems consists of two procedures, the assignment of tasks to each available processor and the ordering of tasks in each processor. The assignment procedure is same with a clustering. The clustering is classified into linear or nonlinear according to the precedence relationship of the tasks in each cluster. The parallel computing system can be modeled with a Directed Acyclic Graph(DAG). By the granularity theory, DAG is categorized into Coarse Grain Type(CDAG) and Fine Grain Type(FDAG). We suggest the linear clustering method for the scheduling of CDAG using the genetic algorithm. The method utilizes a property that the optimal schedule of a CDAG is one of linear clustering. We present the computational comparisons between the suggested method for CDAG and an existing method for the general DAG including CDAG and FDAG.

1. 서론

병렬처리시스템(parallel processing system)에서는 프로그램이 부하 되면 그 프로그램을 먼저 태스크라 하는 작은 조각으로 잘게 나누고, 사용할 프로세서의 수를 결정한 후, 태스크들을 각 프로세서에 할당한다. 그리고 나서, 각 개별 프로세서 내에서, 수행될 태스크들의 순서를 결정하여 프로그램을 수행하게 된다. 이때 하나의 프로그램으로부터 나누어진 태스크들 사이에는 수행의 선후관계에 대한 제약이 있게 되는데, 이들 태스크의 선후관계를 그래프로 표현하면 유방향무환그래프(Directed Acyclic Graph: DAG)가 된다.

한편, 긴급선(후)행 관계에 있는 두 태스크가 서로 다른 프로세서에서 수행되어야할 경우에는 선행태스크가 수행된 프로세서로부터 후행 태스크가 수행될 프로세서로 자료를 이동 시켜야 한다. 따라서 태스크의 선후관계의 제약과 프로세서간 자료 이동시 발생하는 지연 비용을 고려하여 최적의 스케쥴을 작성하는 문제는 매우 어려워진다. 프로세서의 수를 결정하는 문제와 프로세서를 할당하는 문제는 NP-complete 임이, 각 프로세서 내에서 태스크들의 수행 순서를 결정하는 문제는 NP-hard 임이 알려져 있다.[8,15,17,18] 이들 세 가지 문제들 각각에 대하여는 발견적 기법[2,9,16]과 Simulated Annealing[5] 및 유전알고리즘[1] 등이 연구되어 있다. 그러나 이들 각 문제

* 강릉대학교 산업공학과

는 서로 연관되어 있으므로 종합적으로 접근하여 한번에 전체 문제를 동시에 해결되어야 할 것이다.

각 태스크가 처리될 프로세서와, 각 프로세서 내에서 태스크들의 수행순서가 정해지면 전체 프로그램을 수행하는데 소요되는 시간을 알 수 있는데, 이러한 소요시간을 병렬처리시간(Parallel Processing Time: PPT)이라 한다. 여기서 스케쥴링이란, 이러한 프로세서를 할당하고 수행순서를 결정하는 두 가지 문제를 함께 결정하는 것이다.

스케쥴링에 있어서 프로세서 할당을 클러스터링이라고도 하는데, 클러스터에 속한 태스크간 선후관계의 제약조건에 따라 선형 클러스터링과 비선형 클러스터링의 두 가지로 나뉜다.[7] 선형 클러스터링은 클러스터에 속한 태스크들 중 하나를 제외한 모든 태스크가 반드시 하나의 선행 태스크를 가지는 클러스터링이다. 선형 클러스터링이 아닌 클러스터링을 비선형 클러스터링이라 한다. 선형 클러스터링에서는 각 클러스터에 속한 태스크들의 수행순서가 유일하게 결정된다. 그러므로, DAG 의 태스크들을 선형 클러스터링을 찾으면 하나의 스케쥴이 바로 결정된다.

Gerasoulis와 Yang[13]은 DAG에서 자료이동시간과 태스크 수행시간의 상대적 크기를 계산하는 입자이론(Granularity Theory)을 소개하였다. 그들의 입자이론에 의하면, DAG는 Coarse Grain Type DAG(CDAG)와 Fine Grain Type DAG(FDAG)로 구분된다. 그리고 FDAG를 선형 클러스터링하여 생성된 스케쥴이 최적 스케쥴을 포함하고 있다는 보장이 없으나, CDAG의 경우에는 선형 클러스터링에 의하여 생성되는 스케쥴들 중에 최소 스케쥴이 존재한다고 하였다. 즉, CDAG에 있어서 스케쥴링 문제는 선형 클러스터링에 의하여 해결할 수 있게 된다. 그러나 최적의 선형 클러스터링을 찾는 문제도 역시 NP-complete 이다[6,18].

FDAG와 CDAG를 포함하는 General DAG(GDAG) 모델에 대한 스케쥴링 방법으로는 두 단계 이상으로 나누어 계산하는 방법이 많다. 즉, 임의의 선형 클러스터를 만든 후, 그 클러스터들을 묶거나 쪼개는 과정을 일정한 조건이 충족될 때까지 반복함으로써 근사 최적인 클러스터링을 찾는 발견적 기법들이 있다.[4,10,11,12,19] 한편, 박지혁[3] 등은 GDAG 모델에 대한 스케쥴링을 위한 유전알고리즘을 제시한 바 있다.

본 논문에서는 CDAG 모델에 대하여 스케쥴링 문제를 푸는 유전알고리즘을 소개한다. 소개될 유전알고리즘은 선형 클러스터링을 사용하여 PPT가 최소인 스케쥴을 탐색한다. 이때, 사용해야할 프로세서의 개수도 부수적으로 결정하여 준다. 2절에서는 DAG 모델과 스케쥴링 문제에 대해서 자세히 서술한다. 3절에서는 CDAG에 있어서 스케쥴링 문제를 해결하는 유전알고리즘을 소개한다. 그리고 4절에서는 실험을 통하여 제안된 알고리즘의 최적해 탐색 과정을 살펴본다. 그리고 이미 개발된 GDAG 모델에 대한 유전알고리즘과 비교하여 본다.

2. DAG 모델

병렬처리시스템에서 프로그램은 여러 개의 태스크로 분할되는데 이 태스크들 사이에는 선후관계가 존재하게 된다. 여기서 태스크는 개별 처리단위로 긴급선행 태스크를 수행했던 프로세서로부터 필요한 자료를 받아 수행을 시작하면 그 태스크의 수행이 끝나기 전까지는 절대로 중단되지 않는다. 이러한 모델을 병렬계산모델이라 한다.

병렬계산모델은 $G=(V,E,D,\Gamma)$ 인 DAG로 표현된다. 여기서 $V=\{n_1,n_2,\cdots,n_k\}$는 태스크 집합이고, $E=\{e_{ij}|i,j=1,2,\cdots,k\}$는 태스크 i와 j사이의 선후관계 집합이다. $D=\{d_{ij}|i,j=1,2,\cdots,k\}$는 태스크 i와 j사이의 자료이동 시간인데, $e_{ij}\in E$이며 동일 프로세서에서 수행되는 태스크 i와 j에 대해서는 $d_{ij}=0$ 이다. $\Gamma=\{\tau_1,\tau_2,\cdots,\tau_k\}$의 τ_i는 $n_i\in V$인 태스크 i의 수행시간이다.

그림 1에서 (a)는 8개의 태스크를 갖는 DAG의 한 예를 보여준다. 여기서 노드는 태스크를, 호는 선후관계를 의미하며 괄호 안의 숫자는 태스크 노드의 수행시간을, 호 위의 숫자는 태스크간 자료이동시간을 의미한다. 또한, DAG에서 시간집합 $\{D,\Gamma\}\subset G$만을 하나의 행렬 $C=\{c_{ij}|i,j=1,2,\cdots,k\}$로 표현할 수 있는데 그림 1(a)의 DAG를 C 행렬로 표현하면 그림 1(b)가 된다. 그러므로, 그림 1(b)에서 $i\neq j$인 c_{ij}는 d_{ij}를, 그렇지 않은 경우는 τ_i를 의미하게 된다.

병렬처리시스템에서 모든 프로세서가 동일한 수행능력을 갖고 있으며 다른 모든 프로세서와 서로 직접 연결되어 있다고 하자. 또한, 임의의 프로세서에서 처리를 기다리는 태스크가 있고, 그 프로세서가 그 태스크를 처리할

수 있는 상태라면 프로세서는 절대 유휴 상태로 있지 않는다고 하고, 이를 Work-preserving(WP)이라 부르자.

그리고 프로세서의 수가 m개이고 태스크의 수가 n개인 병렬처리시스템의 DAG를 살펴보자. 여기서 하나의 프로세서를 하나의 클러스터로 본다면, DAG에 있어서 스케줄링은 WP를 만족하면서 PPT를 최소화하도록 태스크들을 클러스터링하고 각 클러스터 내에서 각 태스크들의 시작시간을 결정하는 것이다. 이때의 태스크 수행시작시간(t[i,k])과 PPT는 다음 계산을 이용하여 간트 차트로 구할 수 있다.

M: 클러스터들, $|M|=m$

N: 태스크들, $|N|=n$

C: $c_{ij} \in C$, $i=1,\cdots,n$, $j=1,\cdots,n$
$i=j$ 이면 i 태스크의 수행시간, 아니면 i 태스크와 j 태스크 사이의 자료이동시간

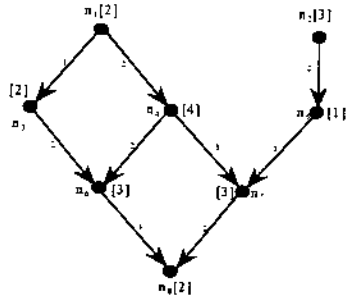
IP[i,j]: i 태스크가 j 태스크의 긴급선행 태스크이면 1, 아니면 0.

T: $t[i,k] \in T$, $i=1,\cdots,n$, $k=1,\cdots,m$
k 프로세서에서 i 태스크의 수행 시작시각

$$t[i,k] = \mathrm{Max}\left\{ \max_{j<i}\{t[j,k]+c_{jj}\}, \max_{\substack{l \\ r\neq k}}\{IP[l,i]\,(t[l,r]+c_{ll}+c_{li})\} \right\}$$

$$PPT = \max_{i}\{t[j,k]+c_{jj}\}$$

t[i, k]는 k 클러스터에서의 i 태스크의 수행시작시간이며 i 태스크와 j 태스크가 동일 클러스터에 할당되는 경우에 자료이동시간 c_{ij}는 0이다. 그러므로, 그림 1(a)모델에서 태스크들의 클러스터 할당이 그림 2(a)의 점선과 같이 되었다면, 이때의 시간 행렬은 그림 2(b)와 같이 된다. 그림 2(a)의 클러스터링은 비선형 클러스터링인 경우이다.

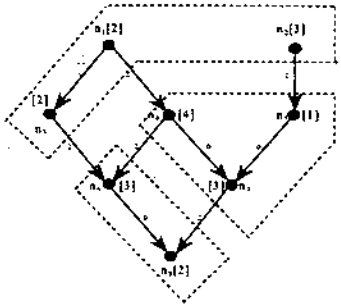


(a) a DAG

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
n_1	2		1	2				
n_2	-	3			2			
n_3	-	-	2			2		
n_4	-	-	-	4		2	3	
n_5	-	-	-	-	1		1	
n_6	-	-	-	-	-	3		1
n_7	-	-	-	-	-	-	3	2
n_8	-	-	-	-	-	-	-	2

(b) C 행렬

그림 1. 8개의 태스크를 갖는 DAG와 시간 행렬



(a) 클러스터가 할당된 DAG

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
n_1	2		0	2				
n_2	-	3			2			
n_3	-	-	2			2		
n_4	-	-	-	4		2	0	
n_5	-	-	-	-	1		0	
n_6	-	-	-	-	-	3		0
n_7	-	-	-	-	-	-	3	2
n_8	-	-	-	-	-	-	-	2

(b) C 행렬

그림 2. 클러스터가 할당된 DAG와 C행렬

따라서 클러스터 내에서의 태스크들의 수행 순서가 하나 이상 존재할 수 있는데, 두 번째 클러스터에서 태스크의 수행 순서는 P_2: $n_4 \rightarrow n_5 \rightarrow n_7$ 또는 P_2: $n5 \rightarrow n4 \rightarrow n_7$ 의 두 가지가 있을 수 있다. 여기서 각 프로세서 내에서의 태스크 순서가 P_1: $n_1 \rightarrow n_2 \rightarrow n_3$, P_2: $n_4 \rightarrow n_5 \rightarrow n_7$, P_3: $n_6 \rightarrow n_8$ 이라면, 위 계산식에 의한 간트 차트는 그림 3과 같이 되고 PPT는 17인 스케줄이 만들어진다.

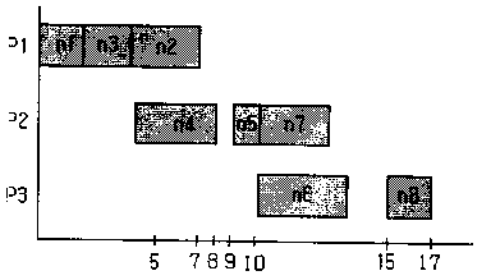


그림 3. 그림2(a) 예의 Gantt 차트

한편 병렬계산 모델에서 태스크의 수행 시간보다 자료 이동시간이 크면, 될수록 적은 수의 프로세서에서 태스크 들을 수행하는 것이 많은 수의 프로세서에서 수행하는 것보다 PPT가 줄어들 수 있다. 그림 4에서, (a)의 DAG의 경우, 태스크 수행시간 ω 보다 자료 이동시간 c가 크면 (b)와 같이 하나의 프로세서에 모든 태스크를 배정하는 것이 스케줄시간 3ω 로 최소 병렬처리시간이 된다. 그러나, 태스크의 수행시간 ω 가 자료 이동시간 c보다 크거나 같은 경우는 (c)와 같이 두 개의 프로세서를 이용하는 것이 $2\omega+c$로 최소 병렬처리시간을 생성한다.

그러므로, 최적의 스케줄을 위해선 병렬계산 모델의 자료이동시간과 태스크 수행시간의 상대적 크기를 살펴볼 필요가 있다. Gerasoulis는 병렬계산모델에서 자료이동시간과 태스크 수행시간의 상대적 크기를 계산하는 입자이론을 소개하였다.[13]

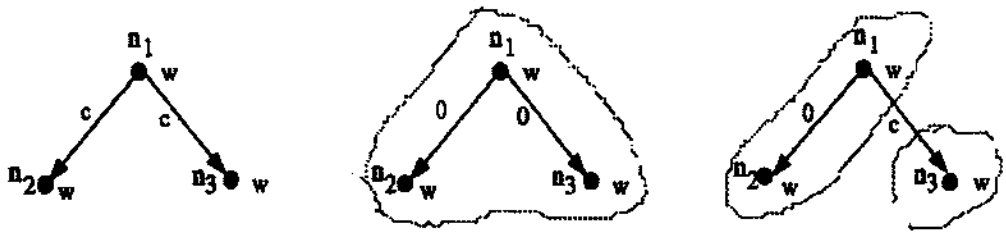
그들은, 다음의 계산 값 $g(G)$ 가 1 보다 크면 CDAG이고, 작으면 FDAG 라고 정의하였다. 아래 식에서 g_x 는 태스크 x의 입자크기를, $g(G)$ 는 모델의 입자성을 나타낸다.

$$g_x = \min\left(\frac{\min_k \{\tau_k\}}{\min_k \{c_{kx}\}}, \frac{\min_k \{\tau_k\}}{\min_k \{c_{xk}\}} \right),$$

$$g(G) = \min_x \{g_x\}$$

그들은 또 FDAG를 선형 클러스터링하여 생성된 스케줄은 최적 스케줄을 포함하고 있다는 보장이 없으나, CDAG의 경우에는 선형 클러스터링에 의하여 생성되는 스케줄들 중에 최소 스케줄이 존재한다는 것을 밝혔다. 즉, CDAG에 있어서 스케줄링 문제는 선형 클러스터링에 의하여 해결할 수 있게 된다.

그리고 또한, 대부분의 병렬계산모델에서는 프로그램을 분할할 때 수행 시간과 자료 이동시간을 적절히 고려함으로써 태스크들이 Coarse Grain Type이 되도록 할 수 있다.[7] 따라서 CDAG에 대해서 선형 클러스터링에 의한 해법이 개발되면 유용하게 사용할 수 있을 것이다.



(a) DAG

(b) $w < c$ 인 경우
PT = $3w$

(c) $w >= c$ 인 경우
PT = $2w + c$

그림 4. 태스크 수행 시간과 자료 이동시간에 따른 프로세서의 수

3. CDAG의 선형 클러스터링을 위한 유전알고리즘

유전알고리즘은 자연계에서 생물 집단이 환경에 적응하면서 진화해 가는 과정을 본떠서 최적해의 탐색에 적용한 것이다. 이 알고리즘은 우수 인자를 갖는 개체일수록 생존 확률이 높게 하고, 생존한 개체들끼리 교배하여 다음 세대의 개체를 생성시키는 확률적 기법이다.[14]

유전알고리즘은 특별히 고안된 부호화 방법을 사용하여, 대상 시스템에서 하나의 상태나 해를 하나의 부호화된 개체로 표현한다. 이들 개체는 인자라고 하는 요소들로 구성된다. 개체들은 그들 각각이 갖는 적합도(Degree of fitness)에 따라 확률적으로 다음 세대의 부모로 선택된다. 선택된 부모들은 교배와 돌연변이를 통하여 자손 개체를 생성하고 일정한 기준이 충족될 때까지 이 과정을 반복한다.

유전알고리즘을 실제 문제에 적용하기 위해선 하나의 스케쥴이 하나의 개체로 표현되도록 하는 부호화 방법, 연산자들 및 각종 파라미터 값들을 설정해야 한다. 연산자로는 교배와 돌변변이를 함께 사용하는데, 그 외의 연산자를 추가로 사용할 수도 있다. 파라미터에는 개체 수, 교배 확률, 돌연변이 확률 등이 있으며 이것 또한, 필요에 따라 다른 파라미터를 사용할 수도 있다.

CDAG에 있어서 선형 클러스터링에 의한 스케쥴링을 위한 유전알고리즘의 부호화 방법, 교배 및 돌연변이 연산자를 다음에 제시한다. 이 연산자들은, 초기 개체들이 선형 클러스터이면 그 연산 결과 생성되는 개체들도 항상 선형 클러스터가 되도록 고안되었다. 따라서 연산 과정 후에 생성된 개체가 선형 클러스터로 유지되도록 하기 위한 별도의 보완 과정이 필요 없는 장점이 있다.

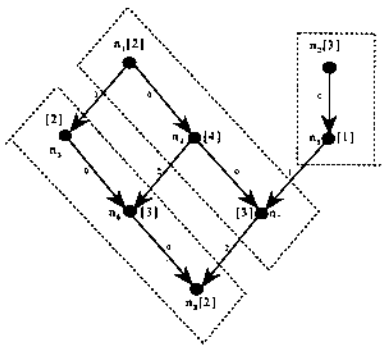
부호화: 아래는 그림 1(a)모델에 대해 개체를 표현한 예이다. 여기에서 개체의 길이는 태스크의 수와 일치하며 i 번째 숫자는 i번째 태스크가 할당되는 클러스터의 번호를 의미한다.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
개체:	1	3	2	1	3	2	1	2*

이 개체는 각 태스크들이 세 개의 프로세서에 할당되어서 각각 {1→4→7}, {3→6→8}, {2→5} 과 같은 순서로 수행된다는 것을 의미한다. 그림 5는 그림 1(a)의 DAG에 위 개체를 표현한 것이다. 소개된 부호화에 의해 표현되는 개체는 하나의 스케쥴과 일대일 대응관계에 있다.

선택 기준: 다음 세대의 자손을 생성하기 위한 부모 개체들은 적합도를 기준으로 선택된다. DAG에서의 스케쥴링문제는 최소 병렬처리시간을 갖는 스케쥴을 구하는 것이 목표이므로 개체가 갖는 스케쥴의 병렬처리시간이 작을수록 적합도는 높아지게 된다. 또한 구해진 적합도에 대해서는 선형비례척도(Linear scale)를 적용한다.

교배 연산자: 교배는 다음의 알고리즘을 따른다. 단계 2에서 해당 태스크가 이미 클러스터를 배정 받고 있는데



(a) 선형 클러스터링 된 DAG

P_1 : n_1 . n_4 . n_7
P_2 : n_3 . n_6 . n_8
P_3 : n_2 . n_5

(b) 태스크 순서

그림 5. 선형 클러스터링된 DAG와 태스크 순서

교배 연산 알고리즘

단계 1. 모든 태스크를 집합 S_1으로 둔다.

단계 2. S_1중 임의의 한 태스크를 선택하여 태스크 T라하자. 부모 1에서, 태스크 T와 같은 클러스터에 할당받은 선행(후행) 태스크들과 부모 2에서, 태스크 T와 같은 클러스터에 할당받은 후행(선행) 태스크들을 자손 1(2)에서 동일한 하나의 클러스터에 할당한다.
이때 자손에서, 태스크 T위치에서 양끝으로 나아가면서 클러스터에 할당하고 이미 클러스터를 배정받고 있는 태스크나 끝 태스크에 도달할 때까지 계속한다. 할당이 끝나면 단계 3으로 간다.

단계 3. 자손 1에서 클러스터에 할당된 태스크를 S_1에서 제외시킨다. S_1에 남아있는 태스크가 하나도 없는 경우는 단계 4로 가고 그렇지 않으면 단계 2로 간다.

단계 4. 자손 2에서 할당받지 않은 태스크를 내림차순으로 정렬하여 집합 S_2로 둔다.

단계 5. S_2의 첫 번째 태스크를 태스크 k라 하자. 자손 2에서 태스크 k의 긴급 선후 태스크들이 할당된 클러스터들을 집합 P_t에 둔다.
자손 2의 열에서 태스크 k와 태스크 k의 긴급 선후 태스크들 사이에 있는 태스크들에 할당된 클러스터들은 P_t에서 제외시킨다. P_t에서 임의의 한 클러스터를 선택하여 태스크 k의 클러스터로 한다. 이때, 집합 P_t가 공집합이면 어느 태스크에도 할당되지 않은 새로운 클러스터에 태스크 k를 할당한다.

단계 6. S_2에서 태스크 k를 제외시킨다. S_2에 남아있는 태스크가 없으면 끝내고, 있으면 단계 5로 간다.

S_1 = { n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8 }
태스크 6 선택.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
P_1	1	3	**2**	1	3	2	1	**2** *
P_2	2	1	2	**3**	1	**3**	1	**3** *
				⇩				
O_1			1			1		1
O_2				1		1		1

(a) 교배과정 1

S_1 = { n_1, n_2, n_4, n_6, n_7 }
태스크 4 선택.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
P_1	1	3	2	**1**	3	2	**1**	2 *
P_2	2	1	2	**3**	1	**3**	1	**3** *
				⇩				
O_1	2		1	2		1		1
O_2				1		1	2	1

(b) 교배과정 2

S_1 = { n_2, n_5, n_7 }
태스크 2선택.

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
P_1	1	**3**	2	1	**3**	2	1	2 *
P_2	2	1	2	**3**	1	3	1	3 *
				⇩				
O_1	2	3	1	2	3	1	3	1
O_2		3		1	3	1	2	1

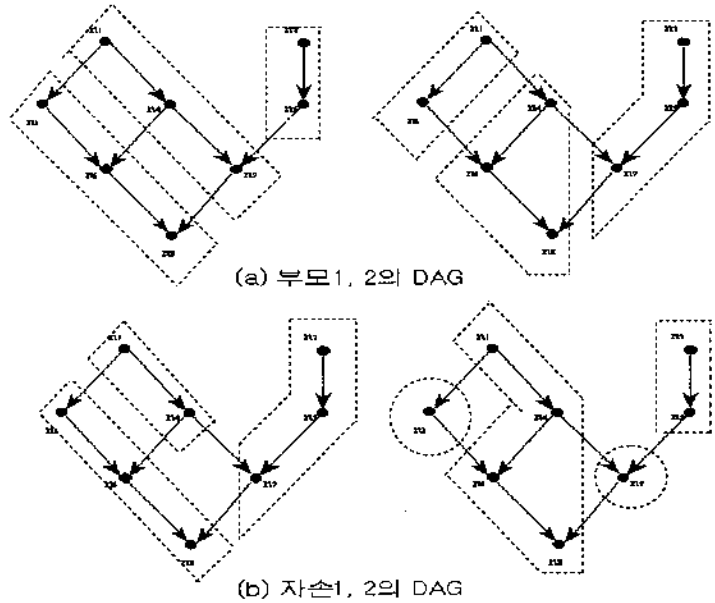
(c) 교배과정 3

S_1 = { }, S_2 = { n_3, n_1 }
P_3 ={ }, P_1 = { 4, 1 }

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
O_1	2	3	1	2	3	1	3	1 *
O_2	*1*	3	*4*	1	3	1	2	1 *

(d) 교배를 통한 자손

그림 6. 교배과정



(a) 부모1, 2의 DAG

(b) 자손1, 2의 DAG

그림 7. 교배에 따른 DAG

도 할당을 계속 진행하면 태스크 열이 끊기어 선형 클러스터가 형성되지 못한다. 단계1, 2, 3에 의하여 선형 클러스터를 만족하는 완전한 자손 1이 생성되며, 자손 2에서 할당받지 않은 태스크들은 단계 4, 5, 6에 의하여 선형 관계를 만족하면서 클러스터에 할당된다.

그림 6은 P_1, P_2 두 개의 부모 개체로 교배하여 자손 O_1, O_2를 생성하는 과정을 순차적으로 보여 준다. 굵은 부분은 선택된 태스크와 자손에 들어가게 되는 인자들을 의미한다. 그림 5(d)에서 S_2={3,1}이 되며, 첫 번째 태스크는 태스크 3이 된다. 태스크 3의 긴급선후 태스크는 태스크 1과 6이며, 이중 클러스터를 배정 받고 있는 태스크는 태스크 6으로 P_3 = {1}이 된다. 하지만, 클러스터 1은 태스크 3과 태스크 6 사이에 있는 태스크 4도 할당받고 있기 때문에, 태스크 3의 클러스터로 지정하면 자손 2는 선형 클러스터를 만족하지 않게 된다. 결국, P_3 = { }이 되므로 태스크 3을 새로운 클러스터 4에 할당한다. 태스크 1은 P_1 ={4,1}이므로 무작위로 선택된 클러스터 1에 할당한다. 그림 7은 이 예를 DAG로 묘사한 것이다. 여기서 *는 선형 클러스터를 만족함을 의미한다.

돌연변이 연산자: 돌연변이는 임의의 한 태스크를 선택하여 긴급선후 태스크들의 클러스터들 중 하나를 임의로 배정한다. 단, 임의로 선택된 태스크가 할당된 클러스터와는 다른 클러스터이어야 하며, 돌연변이에 의해서 끊

초기 클러스터링 방법

단계 0. 모든 태스크의 집합을 S라 둔다. i=1로 둔다.
단계 1. S가 비어 있으면 끝낸다. 아니면 S에서 선행 태스크가 없는 임의의 태스크를 선택하여 태스크 t라 둔다.
단계 2. 태스크 t를 클러스터 P_i에 넣는다.
단계 4. 태스크 t의 후행 태스크들 중의 임의의 하나를 선택하여 태스크 u라 둔다. 태스크 t를 S에서 제외하고 t의 모든 후행 태스크들에 대해서 선행 태스크가 없다고 표시한다.
만약 태스크 t의 후행 태스크가 하나도 없으면 i=i+1로 두고 단계 1로 간다. 아니면 t=u라 두고 단계 2로 간다.

긴 태스크들은 별도의 클러스터에 할당한다. 위의 자손 2를 태스크 6에 대해 돌연변이 시키면 그림 8과 같은 새로운 개체가 생성된다. 이 돌연변이 연산자를 통하여 나온 새로운 개체 역시 선형 클러스터를 만족한다.

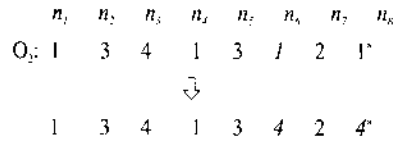


그림 8. 돌연변이 결과

한편, 임의의 초기 선형 클러스터링은 다음의 계산방법을 이용하여 쉽게 구할 수 있다.

4. 전산실험 및 결과

본 절에서는, 이 논문에서 제시한 CDAG 모델인 경우에 적용할 수 있는, 유전알고리즘을 이용한 선형 클러스터링 해법과, 박지혁[3] 등에 의하여 제시된 바 있는 FDAG 모델과 CDAG 모델 모두의 경우에 적용 가능한 유전알고리즘을 전산실험에 의하여 비교하고자 한다. 여기서 전자의 알고리즘을 CGA, 후자의 알고리즘을 GGA라고 표시하기로 한다.

실험대상의 모델들은 C 언어로 작성된 그래프 생성기를 이용하여 무작위로 FDAG 모델과 CDAG 모델을 각각 다섯 개씩 만들었다. 이때 생성기의 파라미터 값은, 층수(number of layer)는 8~10개, 각 층에서의 태스크 수는 3~6개, 폭(width)은 2~5 개, 태스크 수행시간 분포는 U(10,20). 그리고 자료이동 시간 분포는 U(1,6) 와 같이 사용하였다. 그리고 생성된 DAG들은 FDAG의 경우에는 입자성의 크기 $g(G)$가 최소 1 이하이고, CDAG의 경우에는 입자성의 크기 $g(G)$ 가 최소 2 이상이다.

표 2. FDAG 모델에 대한 최소 PPT

문제	GGA	CGA	1-GGA/CGA
F1	119	132	9.8%
F2	86	98	12.24%
F3	95	106	10.38%
F4	62	63	1.59%
F5	74	79	6.33%
평균			8.07%

두 유전알고리즘들 역시 C 언어로 작성하였으며 PC용 Turbo-C 컴파일러를 사용하였다. 두 유전알고리즘에서 사용되는 파라미터 값은 동일하게, 최대 프로세서의 수는 30, 인구수는 30, 교배 비율은 0.4, 돌연변이 비율은 0.05와 같이 적용하였다. 또한 자손 생성을 위한 부모를 선택할 때 $f'=2 \cdot f + 0.1$의 선형비례척도를 사용하며 1000세대까지 실험한다.

우선 CDAG 모델에 대하여 두 유전알고리즘을 적용한 결과는 표 1과 같다. 표 1에 의하면, CGA에 의하여 구

표 1. CDAG 모델에 대한 최소 PPT와 세대수

문제	GGA		CGA		1-CGA/GGA	
	최소 PPT	세대수	최소 PPT	세대수	최소 PPT	세대수
C1	67	732	65	264	3%	63.93%
C2	84	902	86	317	-2.4%	64.86%
C3	71	532	71	248	0%	53.38%
C4	62	824	63	321	1.6%	61.04%
C5	103	549	97	638	5.8%	-16.21%
평균					1.6%	45.4%

해진 해의 PPT가 평균 1.6% 정도 더 좋으며, 수렴 속도는 진행된 세대수를 기준으로 볼 때, 평균 45.5% 정도 더 빠르다. 즉, CDAG 모델의 경우에는 선형클러스터링 해법이 더 빨리 해를 찾는다고 할 수 있다. FDAG 모델에 대하여 두 유전알고리즘을 적용하여 최대한 세대수를 진행시켜 수렴시킨 결과 구해진 최소 PPT는 표 2와 같다. 표 2에 의하면 GGA에 의하여 구해진 해의 PPT가 평균 8.07% 더 좋다. 이는 FDAG모델의 경우에 있어서는 선형 클러스터링이 최적해를 포함하고 있지 않기 때문인 것으로 풀이된다. 그림 9와 10은 각각 임의의 CDAG 모델에 대해서 GGA와 CGA의 해가 수렴되어 가는 과정을 보여주고 있다.

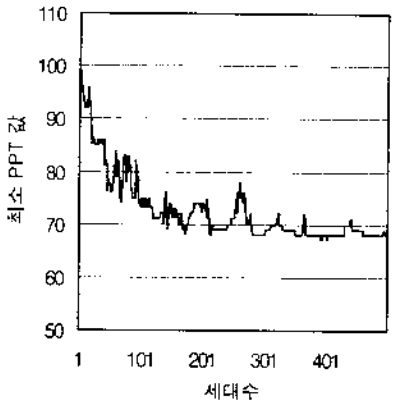


그림 9. GGA의 수행에 따른 문제 C1의 PPT의 수렴

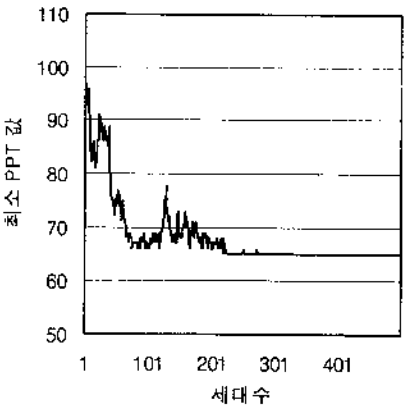


그림 10. CGA 수행에 따른 문제 C1의 PPT의 수렴

5. 결론

대부분의 병렬계산모델에서는 프로그램을 적절히 분할함으로써 태스크들이 Coarse Grain Type이 되도록 할 수 있다. 그렇게 하여 생성된 CDAG 모델의 경우에는 선형 클러스터링을 이용하여 최적 스케쥴을 구할 수 있다. 따라서 본 논문에서는 CDAG 모델의 경우에 유전알고리즘을 이용하여 스케쥴을 구하는 방법을 소개하였다. 이 알고리즘은 주어진 모델의 PPT를 최소로 하는 선형 클러스터를 탐색한다. 또한 이 유전알고리즘에서 사용한 부호화, 교배 및 돌연변이 연산자 모두가 선형 클러스터를 유지하도록 고안되었다. 즉, 초기 개체들이 선형 클러스터이면 그 연산 결과 생성되는 개체들도 항상 선형 클러스터가 되도록 고안되었다. 따라서 연산 과정 후에 생성된 개체가 선형 클러스터로 유지되도록 하기 위한 별도의 보완 과정이 필요 없는 장점이 있다. 또한 이 유전알고리즘은 최소의 PPT를 갖는 스케쥴을 탐색함과 동시에 최적 프로세서의 개수도 부수적으로 구하여 준다.

제안된 알고리즘은 유방향무환네트워크 구조에서 각 노드의 가중치가 호의 가중치보다 상대적으로 큰 유형의 문제들을 푸는데 응용할 수 있을 것으로 기대된다. 그러나, 각 노드의 가중치가 호의 가중치보다 상대적으로 작아서 CDAG 형태로 모델링 할 수 없는 경우에는 제안된 알고리즘으로 최적해를 찾을 수는 없다. 또한 제안한 알고리즘과, 기존의 일반적인 DAG에 적용할 수 있는 유전알고리즘을 비교하여 전산 실험한 결과, FDAG 모델인 경우에는 제안된 알고리즘의 해가 좋지 않으나, CDAG 모델에 있어서는 제안된 알고리즘이 더 좋은 해를 더 빨리 찾는다.

참고문헌

[1] 김의창, 홍영식, "다중처리기 시스템에서의 태스크할당을 위한 유전알고리즘", 정보과학회지, Vol. 20, No. 1, pp.43-5, January, 1993.

[2] 류재철, "N-큐브 다중프로세서 시스템에서 태스크 할당 기법", 정보과학회지, Vol. 20, No. 5, pp.739-750, May, 1993.

[3] 박자혁, 성기석, "프로세서의 수가 한정되어있는 병

렬계산 모델에서 유전알고리즘을 이용한 스케줄링 해법", 한국경영과학회지, Vol. 23, No 2, 1998.

[4] Adam T., Chandy K.M. and Dickson J.R., "A comparison of list schedules for parallel processing Systems", *CACM*, Vol. 17(22), pp.685-690, 1974.

[5] Bollinger, S.W. and Midkiff, S.F., "Processor and Link Assignment in Multicomputers using Simulated Annealing", *Proceeding of the 1988 Int. Conf. on Parallel Processing*, Vol. 1, pp.1-7, Aug., 1988.

[6] Chretienne P., "A polynomial algorithm to optimally schedule tasks over an ideal distributed system under tree-like precedence constraints", *Eur. J. Oper. Res.*, 2, pp.225-230, 1989.

[7] Chretienne *et al.*, *Scheduling Theory and its Applications*, John Wiley & Sons, 1995.

[8] Coffman P., and Denning P.J., *Operations Systems Theory*, Prentice-Hall, Engflewood Cliffs, NJ, 1973.

[9] El-Rewini H. and Lewis T.G., "Scheduling parallel program tasks onto arbitrary target machines", *J. Parallel Distrib. Comput.*, 9, pp.138-153, 1990.

[10] Gerasoulis A. Jiao J. and Yang T., *A Multistage Approach for Scheduling Task Graphs on Parallel Machines*, DIMACS series, American Mathematical Society, 1995.

[11] Gerasoulis A. and Nelken I., "Static scheduling for linear algebra DAGs", *Proceedings of HCCA*, 4, pp. 671-674, 1989.

[12] Gerasoulis A. and Yang T., "A comparison of clustering heuristic for scheduling DAGs on multiprocessors", *J. Parallel Distrib. Comput.*, 16, pp. 276-291, 1992.

[13] Gerasoulis A. and Yang T., "On the granularity and clustering of directed acyclic task graphs", *IEEE Trans. Parallel Distrib. Syst.*, 4, pp.686-701, 1993.

[14] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, NY, 1989.

[15] Hoogeveen J.A., Van de Velde S.L. and Veltman B., "Complexity of scheduling multiprocessor tasks with prespecified processor allocations", *CWI Report BS-R9211*, Amsterdam.

[16] Lee, Y.S. and Aggarwal, J.K., "A Mapping Strategy for Parallel Processing", *IEEE Trans. on Computer*, Vol. C-36, No. 4, pp.432-442, April, 1987.

[17] Lenstra J. K. and Rinnooy Kan A. H. G., "Complexity of scheduling under precedence constraints", *Oper. Res.*, 26, 1978.

[18] Papadimitriou C. and Yannakakis M., "Towards on an architecture-independent analysis of parallel algorithms", *SIAM J. Comput.*, 19, pp.322-328, 1990.

[19] Yang T. and Gerasoulis A., "DSC: Scheduling Parallel Tasks on Unbounded Number of Processors", *IEEE Trans. on Parallel and Distributed System.*

98년 3월 최초 접수, 98년 9월 최종 수정