韓國經營科學會誌
第23卷 第2號
1998年 6月

프로세서의 수가 한정되어있는 병렬계산모델에서 유전알고리즘을 이용한 스케쥴링해법

성기석* · 박지혁*

A Scheduling Method on Parallel Computation Models with Limited Number of Processors Using Genetic Algorithms

Kiseok Sung* · Jeehyuk Park*

Abstracts

In the parallel processing systems, a compiler partitions a loaded program into tasks, allocates the tasks on multiple processors and schedules the tasks on each allocated processor. In this paper we suggest a Genetic Algorithm(GA) based scheduling method to find an optimal allocation and sequence of tasks on each processor. The suggested method uses a chromosome which consists of task sequence and binary string that represent the number and order of tasks on each processor respectively. Two correction algorithms are used to maintain precedency constraints of the tasks in the chromosome. This scheduling method determines the optimal number of processors within limited numbers, and then finds the optimal schedule for each processor. A result from computational experiment of the suggested method is given.

1. 서 론

다중 프로세서로 구성된 병렬처리 시스템은 부하되는 프로그램의 수행 속도를 증진시키기 위해서 개발되었다. 이를 위하여 컴파일러는 프로그램이 부하되면 부하된 프로그램을 실행하기 이전에 다음의 네 가지 과정을 순차적으로 실행하게 된다.[7,8]

(1) 프로그램을 작은 조각(Task)으로 나누는

* 강릉대학교 산업공학과

분할

(2) 사용할 프로세서 수의 결정

(3) 태스크들의 프로세서로의 배정

(4) 각 프로세서 내에서의 태스크들의 순서 결정

분할에 의해 나누어진 프로그램의 조각을 태스크라 하며, 각 단계의 목적은 부하 된 프로그램의 병렬처리 시간 (PPT : Parallel Processing Time)을 최소화하는 것이다.

프로그램의 분할이 완료되면 태스크간에는 선후 관계(precedency)가 존재하게 되며 이들을 수행하는 프로세서간에는 자료 이동이 발생한다. 이러한 프로세서간의 자료 이동의 필요성 때문에 프로그램을 분할하고 난 후 컴파일러가 수행해야할 세 가지 후속 문제들을 해결하는데 어려움이 있다. 프로세서 수를 결정하는 문제와 태스크를 배정하는 문제는 NP-complete 이고, 각 프로세서 내에서의 태스크들의 실행 순서를 결정하는 문제는 NP-hard 임이 알려져 있다.[8,18,23,24]

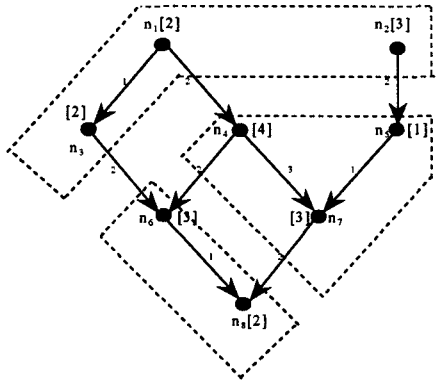
그 동안 이러한 세 가지 문제 각각에 대해서는 발견적 기법들이 고안되어 있다.[2,7,11,21] 특히, 태스크 배정 문제에 대해서는 발견적 기법뿐만 아니라 Simulated annealing이나 유전알고리즘에 의한 해법에 대해서도 연구되어져 왔었다.[1,4,10,22] 병렬처리 시스템에서 스케줄링 문제는 태스크 배정 문제와 태스크 순서 결정 문제를 포함한다. 병렬처리 시스템에서의 스케줄링을 위한 발견적 기법들 또한 제안되었는데, 이 기법들은, 프로세서가 무한개이며 각 프로세서들은 모두 연결되어 있다고 가정하였다.[12,21,26,27,28,29,30] 한편, 분할된 프로그램은 태스크들의 입자성에 의하여 Coarse Grain (CG)과 Fine Grain(FG)의 두 가지 형태로 나뉘어진다. 지금까지 연구되어 온 알고리즘들은 분할된 프로그램이 어떤 형태를 갖는지에 따라서 그 수행도가 차이가 난다.[3,7,13,14,24,28]

본 논문에서는 병렬처리 시스템의 스케줄링을 해결하기 위한 유전알고리즘을 소개한다. 소개할 알고리즘은 주어진 모델의 형태에 관계없이 일정한 수준의 수행도를 갖고 있다. 또한 최소 스케줄을 찾는 동안 가용한 프로세서의 수 이내에서의 최적 프로세서수도 결정하여 준다. 유전알고리즘은 Job shop 스케줄링이나 외판원 문제와 같은 조합최적화 문제의 해법에 효율적이라고 알려져 있다.[5,9,16,19,20,21,25,31]

2 장에서는 병렬처리 시스템에서의 스케줄링 문제에 대한 기본 용어 및 가설을 설명한다. 3 장에서는 병렬처리 시스템에서 프로그램이 부하되었을 때 최소 스케줄을 찾기 위한 유전알고리즘을 소개한다. 그리고 4 장에서는 전산실험을 통하여 제안된 알고리즘의 최소 스케줄 탐색 과정을 살펴보고 분할된 프로그램의 형태에 따라 수행도가 얼마나 차이가 나는지를 알아본다.

2. 문제 정의

병렬처리 시스템에서 프로그램이 부하되면 여러 개의 태스크로 분할되는데 이 태스크들 사이에는 선후 관계가 존재하게 된다. 이러한 모델을 병렬계산 모델(Parallel Computation Model)이라 한다. 여기서 태스크는 개별 처리 단위로 긴급 선행 태스크를 수행했던 프로세서로부터 필요한 자료를 받아 수행을 시작하면 그 태스크의 수행이 끝나기 전까지는 절대로 중단



(a) a DAG

$$
\begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \\ n_7 \\ n_8 \end{array}
\begin{array}{c}
\begin{array}{cccccccc} n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 & n_8 \end{array} \\
\begin{bmatrix}
2 & & 0 & 2 & & & & \\
- & 3 & & & 2 & & & \\
- & - & 2 & & & 2 & & \\
- & - & - & 4 & & 2 & 0 & \\
- & - & - & - & 1 & & 0 & \\
- & - & - & - & - & 3 & & 0 \\
- & - & - & - & - & - & 3 & 2 \\
- & - & - & - & - & - & - & 2
\end{bmatrix}
\end{array}
$$

(b) C행렬

[그림 1] 8개의 태스크를 갖는 DAG와 시간 행렬

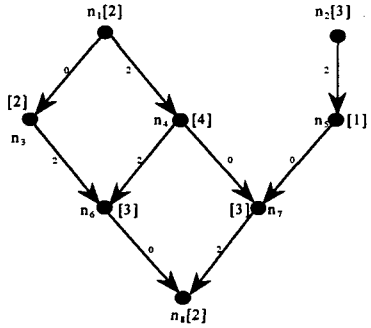
되지 않는다.

병렬계산 모델은 $G=(V, E, D, \Gamma)$ 인 DAG(Directed Acyclic weighted task Graph)로 표현된다. 여기서 $V=\{n_1, n_2, \ldots, n_k\}$는 태스크 집합이고, $E=\{e_{i,j}/i,j=1, 2,\cdots, k\}$는 태스크 i 와 j 사이의 선후 관계 집합이다. $D=\{d_{i,j}/i,j=1, 2, \cdots, k\}$ 는 태스크 i 와 j 사이의 자료 이동(communication delay) 시간이며, $e_{i,j}\in E$ 인 태스크 i 와 j 가 동일 프로세서에서 수행된다면 $d_{i,j}=0$ 이 된다. $\Gamma=\{\tau_1, \tau_2,\cdots, \tau_k\}$ 의 τ_1 는 $n_1\in V$ 인 태스크 i 의 수행시간(computation time)이다.

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

프로세서의 수가 m개인 병렬처리시스템에 n개의 태스크로 이루어진 병렬계산모델을 살펴보자. 모든 프로세서가 동일한 수행능력을 갖고 있으며 다른 모든 프로세서와 상호 연결되어 있다고 하자. 즉, 태스크의 처리 시간과 태스크간 자료 이동시간은 어떤 프로세서에서 처리되고 이동되는지와 관계없이 항상 일정하게 된다. 또한, 프로세서에 할당된 태스크중 처리를 기다리는 태스크가 있고 동시에 프로세서가 그 태스크를 처리할 수 있는 상태라면 프로세서는 결코 유휴 상태에 있지 않는다고 하자 (work-preserving).

이러한 조건하에서, 모든 태스크를 각 프로세서에 배정하고 배정된 프로세서 내에서의 태스크 순서를 결정하면 하나의 스케줄이 생성되어 각 태스크들의 수행시작 시간을 알 수 있게 된다. 그러므로, 병렬계산 모델에서 스케줄링은 work-preserving을 만족하면서 병렬처리 시간(PPT:Parallel Processing Time)을 최소화하는



(a) 프로세서가 배정된 DAG

$$C = \begin{array}{c} \\ n_1 \\ n_2 \\ n_3 \\ n_4 \\ n_5 \\ n_6 \\ n_7 \\ n_8 \end{array} \begin{array}{c} \begin{array}{cccccccc} n_1 & n_2 & n_3 & n_4 & n_5 & n_6 & n_7 & n_8 \end{array} \\ \begin{bmatrix} 2 & & 0 & 2 & & & & \\ - & 3 & & & 2 & & & \\ - & - & 2 & & & 2 & & \\ - & - & - & 4 & & 2 & 0 & \\ - & - & - & - & 1 & & 0 & \\ - & - & - & - & - & 3 & & 0 \\ - & - & - & - & - & - & 3 & 2 \\ - & - & - & - & - & - & - & 2 \end{bmatrix} \end{array}$$

(b) C행렬

[그림 2] 프로세서가 배정된 DAG와 C행렬

각 태스크의 각 프로세서로의 배정과 프로세서 내에서의 각 태스크들의 시작 시간 결정을 의미한다. 이때의 태스크 수행시작 시간 ($t[i,k]$)과 병렬처리 시간(PPT)은 다음 계산을 이용하여 Gantt 차트로 구할 수 있다.

M : 프로세서들, $|M|=m$

N : 태스크들, $|N|=n$

C : $c_{i,j} \in C$, $i=1,\cdots,n$, $j=1,\cdots,n$

태스크의 수행시간 iff $i=j$

태스크 i와 j사이의 자료 이동시간 iff $i \neq j$

$IP[i,j]$: 1 iff j의 긴급선행(imminent precedent) 태스크가 i

0 iff 기타

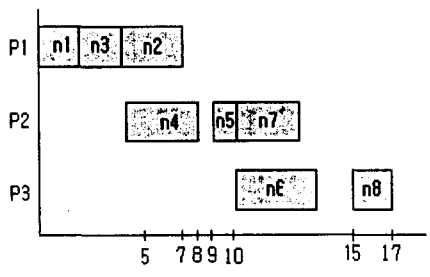
T : $t[i,k] \in T$, $i=1,\cdots,n$, $k=1,\cdots,m$

work-preserving을 만족하는 k 프로세서에서 i 태스크의 수행 시작시각

$$t[i,k] = \max\{t[j,k] + c_{ji},\ IP[l,i] \cdot (t[l,r] + c_{ll} + c_{li})\},\ k \neq r$$

$$PPT = \max\{t[i,k] + c_{ii}\}$$

$t[i,k]$는 k 프로세서에서의 i 태스크의 수행시작시간이며 c_{ij} 는 i 와 j 태스크가 동일 프로세서에 배정되는 경우 0의 자료 이동시간이 된다. 그러므로, 그림 1 의 (a) 모델에서 태스크들의 프로세서 배정이 그림 2 의 (a)의 점선과 같이 되었다면, 이때의 시간 행렬은 그림 2 의 (b)와 같이 된다. 그림 2 의 (a)의 예에서 각 프로세서 내에서의 태스크 순서가 $P_1 : n_1 \rightarrow n_3 \rightarrow n_2$, $P_2 : n_4 \rightarrow n_5 \rightarrow n_7$, $P_3 : n_6 \rightarrow n_8$ 이면, 위 계산식에 의한 Gantt 차트는 그림 3과 같이 되고 병렬처리 시간이 17 인 스케줄이 만들어진다.



[그림 3] 그림2의 (a) 예의 Gantt 차트

3. 병렬계산 모델에서의 스케쥴링을 위한 유전알고리즘

유전알고리즘은 생물 집단의 진화 과정을 최적해의 탐색에 적용한 것이다. 이 알고리즘은 우수 인자를 갖는 개체일수록 생존 확률을 높도록 하여 생존한 인자들끼리 다음 세대를 생성시키는 확률적 기법이다.[15,31]

유전알고리즘은 일반적으로 특별히 고안된 부호화 방법(encoding scheme)을 통하여 하나의 해가 하나의 열(string)이 되도록 표현하는데 부호화를 통하여 나온 열을 개체라 하고 개체의 요소(element)를 인자(gene)라 한다. 부호화된 개체들은 각 개체가 갖는 적합도(degree of fitness)에 따라 확률적으로 다음 세대의 부모로 선택된다(selection and reproduction). 선택된 부모들은 교배(crossover)와 돌연변이(mutation)를 통하여 자손을 생성하고 일정 기준이 충족될 때까지 이 과정을 반복한다.[15,31] 이 유전알고리즘은 John Holland(1975)에 의해서 처음 소개된 이후로 많은 응용분야에 매우 효율적으로 적용되어 왔다.[5,9,16,17,19,20,25] 이러한 유전알고리즘을 실제 문제에 적용하기 위해선 다음의 네 가지가 기본적으로 해결되어야 한다.

(1) 해를 적절한 개체로 표현하는 부호화
(2) 선택 및 복제
(3) 교배 연산자
(4) 돌연변이 연산자

본 논문에서 제안하는 병렬계산 모델의 스케쥴링에 사용할 유전알고리즘은 위의 네 가지 기본적으로 해결되어야 할 사항들을 다루고 있다. *3.1* 절에서 병렬계산모델에 있어서의 스케쥴링을 위한 유전알고리즘의 부호화 방법을 소개한다. 또한, *3.2* 절에서는 이와 같은 부호화 방법으로 표현된 개체가 실행가능성(feasibility)을 유지하기 위해서 적용하는 과정에 대해서 설명한다. *3.3* 절과 *3.4* 절에서는 제안된 유전알고리즘이 사용하는 선택 및 복제 방법, 그리고 교배연산자 및 돌연변이 연산자들에 대해서 설명한다.

3.1 부호화

여기서 개체는 두 부분으로 구성된다. 한 부분은 태스크들의 순서를 의미하며 다른 하나는 각 프로세서에 할당되는 태스크의 수를 이진열로써 표현한다. 태스크가 n개, 프로세서가 m개이면, 모든 태스크의 개수가 표현 가능하도록 ($2^a \geq n$) 단위길이 a를 프로세서의 수만큼 나열하므로 한 개체의 전체 길이는 $n + a \cdot m$이 된다. 아래는 8 개의 태스크와 4 개의 프로세서가 있는 경우의 개체의 예이다.

```
3 4 1 5 2 7 6 8 011 011 010 101
↪task sequence↩ ↓   ↓   ↓   ↓
                3   3   2   4
```

이 개체의 전체 길이는 8 + 3 · 4 = 20이며 이진열 부분은 프로세서 P_1, P_2, P_3, P_4 각각에 배정될 태스크수가 3, 3, 2, 4 개임을 의미한다. 또한, 태스크 번호 *{3, 4, 1}*, *{5, 2, 7}*, *{6, 8}* 이 각각의 프로세서 P_1, P_2, P_3 에 배정되고 각 프로세서 내에서의 태스크 순서는 *3→4→1*, *5→2→7*, *6→8* 이 된다. 이때 태스크가 모두 8 개이므로 어떤 태스크도 배정받지 못한 나머지 하나의 프로세서는 유휴 상태가 된다. 이 부호화 방법에 의해 생성된 개체는 태스크들이 수행되어

야 할 프로세서와 각 프로세서 내에서의 태스크 순서에 대한 정보를 모두 담고 있다.

3.2 실행가능성 유지

앞 절에서 소개한 부호화에 의한 개체는 병렬계산모델의 선후 관계 제약에 위배될 수 있으며 이런 경우 병렬처리시간의 계산이 불가능하게 된다. 따라서 실행가능성을 만족시키기 위해서, Gantt 차트를 이용하여 각 개체의 병렬처리 시간을 계산하는 과정에서 개체를 수정을 한다. 모든 개체는 다음 *알고리즘 1* 을 거친다. *알고리즘 1* 은 각 프로세서 내에서의 태스크들이 선후 관계를 만족하는지 판별하고 만족하지 않는 경우 고유 정보는 유지시키면서 각 프로세서내의 태스크간 선후 관계를 만족하도록 개체를 수정한다.

주어진 DAG가 그림 1 의 (a)와 같을 때, 부호화에서 예로 소개한 개체는 사용중인 프로세서수가 3 개이므로 $NP = 3$ 이 되고 단계 2 에 의하여 $PS_1 = \{3, 4, 1\}$ 이 된다. 단계 3 에서 태스크 4 의 후속 태스크 중에는 태스크 3 이

알고리즘 1

단계 1. 사용중인 프로세서의 수를 NP, I=1 로 놓는다.

단계 2. 프로세서 i 에 배정된 태스크들을 수행순서대로 집합 PS_i에 놓고, $k = 1$, $j = 2$ 로 둔다.

단계 3. 집합 PS_i 의 j 번째 태스크를 태스크 T 라 하자. 집합 PS_i 의 k 번째 태스크가 태스크 T 의 후속 태스크들 중에 존재하면 집합 PS_i 에서 $k \sim j-1$ 태스크들을 태스크 T 의 바로 뒤로 옮긴 후, 단계 5 로 가고 존재하지 않으면 단계 4 로 간다.
여기서, 태스크 T 의 후속 태스크들은 모델의 시간 행렬에 나타나 있다.

단계 4. $k = j-1$ 이면 단계 5 로 가고 그렇지 않으면 ($k < j-1$), $k = k+1$ 로하고 단계 3 으로 간다.

단계 5. j번째 태스크가 집합 PS_i 의 마지막 태스크이면 단계 6 으로 가고 그렇지 않으면 $j = j+1$, $k = 1$ 로 놓고 단계 3 으로 간다.

단계 6. $i < NP$ 이면 $i = I+1$ 로 하여 단계 2 로 가고 $i = NP$ 이면 끝낸다.

존재하지 않고 단계 4, 5 에서는 $k=j-1$ 이면서 마지막 태스크가 아니므로 세 번째 태스크에 대하여 단계 3 을 다시 수행하게 된다. 이때 PS_1의 세 번째 태스크 1 은 첫 번째 태스크 3 을 후속 태스크로 갖고 있으므로 집합 PS_1 안에서의 태스크 순서는 *{1, 3, 4}* 가 된다. 이러한 방법으로 프로세서 2, 3 에 대해서도 태스크 순서를 수정하면 개체는 아래와 같이 된다. 이 개체는 각 프로세서 내에서의 태스크간 선후 관계 제약을 모두 만족하고 있다.

1 3 4 | 2 5 7 | 6 8 | 011 011 010 101*

여기서 *는 실행가능성을 만족한다는 것을 의미한다. *알고리즘 1* 은 프로세서 내에서의 태스크 선후 관계만을 고려한 것으로 프로세서간의 태스크 선후 관계는 만족하지 않을 수도 있다. 이것은 선후 관계가 존재하지 않는 둘 이상의 태스크 군들이 동일 프로세서에 배정되었을 때 나타날 수 있으며, Gantt 차트를 이용하여 병렬처리시간을 계산하는 중간에 발생한다. 이 경우는 그때까지 수행 시작 시간이 결정되지 않은 나머지 태스크들을 대상으로 *알고리즘 2* 를 적용한다.

그림 4 는 그림 1 의 (a)의 DAG에 대한 어느 한 개체에 *알고리즘 2* 를 적용한 예이다. *알고리즘 2* 를 적용하기 이전 개체의 태스크순서는 4→8, 3→6→2, 5→1→7로 각 프로세서 내에서의 태스크간 선후 관계는 만족하지만, 다른 프로세서에 있는 태스크와의 선후 관계는 만족

알고리즘 2

단계 1. 수행되어야 할 태스크가 남아있는 프로세서의 수를 *NP*로, $i = 1$ 로 놓는다.

단계 2. 프로세서 i에 수행을 기다리는 태스크들을 수행순서대로 집합 PS_i 에, $j = 2$ 로 놓는다.

단계 3. 집합 PS_i 에서 j 번째 태스크가 첫 번째 태스크의 후속 태스크가 아니면 $1 \sim (j - 1)$ 태스크들을 j 번째 태스크 바로 뒤로 옮긴 후 단계 5로 가고 후속 태스크일 때는 단계 4로 간다.

여기서, 첫 번째 태스크의 후속 태스크들은 모델의 시간 행렬에 나타나있다.

단계 4. 집합 PS_i 에서 j 번째 태스크가 마지막 태스크이면 단계 5 로가고 그렇지 않으면, $j = j+1$로하고 단계 3으로 간다.

단계 5. $i = NP$ 이면 끝내고 그렇지 않으면 $i = I+1$ 로 놓은 후 단계 2 로 간다.

하지 않기 때문에 실행이 불가능하다. *알고리즘 2* 의 단계 3 은 프로세서 내에서 선후 관계가 존재하지 않은 태스크를 찾아 수행 순서를 바꾸어 주는 과정으로 집합 PS_2의 태스크 순서 3→6→2를 2→3→6으로, 집합 PS_3의 태스크 순서 5→1→7 은 1→5→7 로 수정한다. 태스크 3, 6 과 태스크 2 그리고 태스크 5, 7과 태스크 1 각각은 각 프로세서 내에서 선후 관계가 존재하지 않기 때문에 이 알고리즘에 의해 수정된 다음에도 프로세서 내에서의 선후관계는 만족한다. 고안된 두 가지의 알고리즘을 적용하면 Gantt 차트를 이용하여 모든 개체의 병렬처리 시간을 구할 수 있다.

4 8 | 3 6 2 | 5 1 7 | 010 011 111 001

⇓

4 8 | 2 3 6 | 1 5 7 | 010 011 111 001*

[그림 4] 알고리즘 2 의 수행

3.3 선택기준

다음 세대의 자손을 생성하기 위한 부모 개체들은 적합도를 기준으로 선택된다. 병렬계산 모델에서의 스케쥴링문제는 최소 병렬처리시간을 갖는 스케쥴을 구하는 것이 목표이므로 개체가 갖는 스케쥴의 병렬처리시간이 작을수록 적합도는 높아지게 된다. 병렬처리시간은 앞에서 소개한 Gantt 차트를 이용하여 구한다. 이때 개체가 가능성을 만족하지 못하면 소개된 두 가지 알고리즘을 적용하게 된다. 유전알고리즘은 세대를 진행함에 있어서 초기에는 적합도의 심한 차이로 인하여 조기 근사를, 후반에는 적합도의 미세차로 인하여 Random walk을 하는 경향이 있다.[15] 이러한 단점을 극복하기 위하여 부모를 선택할 때 기존의 적합도에 대해 Linear scaling을 적용한다.

3.4 연산자

부모가 선택되면 교배와 돌연변이 과정을 통해서 자손을 생성하게 된다. 교배는 개체의 태스크열과 이진열의 두 부분을 따로따로 수행한다. 태스크열 부분에는 순서결정문제(Sequencing Problem)에 많이 쓰이는 Partially Mapped Crossover(PMX), Order Crossover(OX) 및 Cycle Crossover(CX)중 OX 연산자를, 이진열 부분에는 Swap 연산자를 사용한다. 그림 5 는 부모 개체 P_1, P_2에 OX와 Swap 연산자를 각각의 부분에 사용하여 자손 O_1, O_2를 생성하는 모습이다.

P_1 : 34 | 152 | 768 011 011 010 101 *

× × ↕

P_2 : 36 | 827 | 154 110 100 011 001 *

⇓

O_1 : 87 | 152 | 436 011 000 011 001

O_2 : 15 | 827 | 634 110 111 010 101

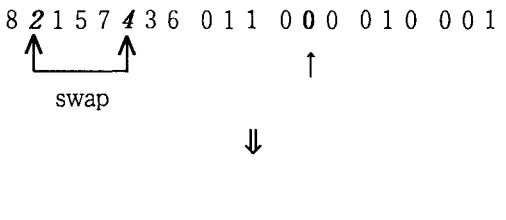
[그림 5] OX와 Swap 연산자를 이용한 교배

유전알고리즘의 수행에 있어서 돌연변이는 새로운 해공간을 탐색할 수 있도록 하는 중요한 역할을 한다. 태스크열 부분에는 임의로 선택한 두 태스크의 위치를 서로 바꾸는 Swap 연산자

를, 이진열 부분에는 임의의 한 인자를 선택하여 0은 1로, 1은 0으로 바꾸는 Transformation 연산자를 사용한다. 그림 6 은 대상 개체에 Swap과 Transformation 연산자를 이용하여 돌연변이를 수행한 예이다.

여기서 자손 개체들이 의미하는 프로세서의 수는 부모개체들이 사용했던 프로세서의 수와 다를 수 있다. 이것은 사용 할 프로세서의 수를 결정하는 개체의 이진열 부분이 교배 및 돌연변이 연산자에 의하여 바뀌기 때문이다. 그러므로, 소개된 유전 알고리즘은 최소 병렬처리 시간을 갖는 스케쥴을 찾는 동안 최적 프로세서의 수 또한 동시에 탐색하게 된다.

4. 전산 실험 및 결과



[그림 6] Swap과 Transformation을 이용한 돌연변이

4.1 실험 모델

본 절에서는 전산 실험을 통하여 소개된 유전알고리즘이 Coarse Grain(CG) 및 Fine Grain (FG) 형태의 병렬계산 모델에서 최적해를 탐색하는 지를 알아보았다. 실험 대상의 병렬계산 모델과 유전알고리즘 모두는 C 언어로 프로그래밍 하였고 PC 용 Turbo-C 컴파일러를 사용하였다. 사용한 병렬계산 모델들은 3~6 개의 태스크들이 한 층(layer)을 이루며 8~10개의 층을 갖도록 하였으며, 한 태스크가 가지는 후속 태스크가 2~5 개가 되도록 하였다. CG와 FG형태의 병렬계산 모델 각각을 5 개씩 랜덤하게 만들어 실험하였다. 표 1 은 생성된 모델들의 단위 태스크 수행 시간과 자료 이동시간의 범위를 보여주고 있다.

〈표 1〉 실험모델들의 시간범위

해당모델	태스크 수행시간	자료이동시간
FG 모델들	1~5	3~10
CG 모델들	10~20	1~6

4.2 실험 결과

유전알고리즘의 파라미터값들은 다음과 같이 적용하였다.

프로세서수(Number of Processors) : 30
집단의 크기(Population Size) : 30
교배율(Crossover rate) : 0.4
돌연변이율(Mutation rate) : 0.05

소개된 유전알고리즘은 병렬처리 시간을 최소로 하는 최적 프로세서의 수를 최적 스케쥴을 찾는 동안 구하고 최적 프로세서의 수가 사용가능한 프로세서의 수를 넘지 않도록 설계되었으므로 사용 가능한 프로세서의 숫자만을 정해주면 된다. 자손을 생성하기 위해서 부모를 선택할 때, $f' = 2 \cdot f + 0.1$의 Linear Scaling을 사용하며 1000 세대까지 실험하였다.

〈표 2〉 CG와 FG형태의 병렬계산모델에 대한 실험결과

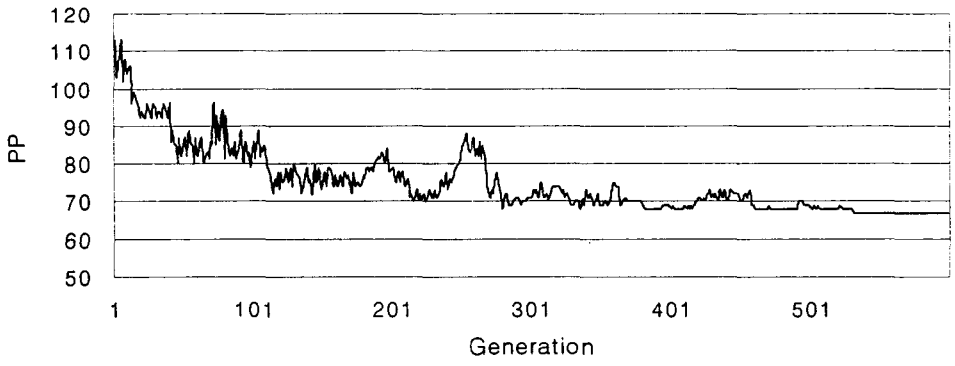
CG 모델	세대 번호	FG모델	세대 번호
C-1 모델	532	F-1 모델	821
C-2 모델	902	F-2 모델	639
C-3 모델	736	F-3 모델	702
C-4 모델	824	F-4 모델	598
C-5 모델	549	F-5 모델	879
평균	708.6	평균	727.8

10개의 병렬계산 모델에 소개한 알고리즘을 적용한 결과가 표 2 에 나타나 있다. 표 안의 숫자는 구해진 병렬처리 시간 중 최소인 시간을 찾은 세대번호이다. 표 2 는 CG와 FG 형태의 병렬계산 모델에 대하여 소개한 유전알고리즘이 비슷한 수행도를 갖는다는 것을 보여준다. 그림 6 은 C-1 모델에 대해서 소개한 유전알고리즘이 최소 병렬처리 시간을 찾아가는 과정을 보여준다. 이 실험에서 소개된 알고리즘이 최소 병렬처리 시간을 탐색하며 병렬계산 모델의 종류와 관계없이 균일한 수행도를 보여주는 것으로 나타났다.

5. 결 론

병렬계산 모델의 스케쥴링 문제의 해결을 위해 제시된 지금까지의 방법은 발견적 기법들이었으며 그 외의 방법들은 프로세서로의 태스크 할당과 같은 스케쥴링 문제의 일부분에 국한되어 왔다. 본 연구에서는 병렬계산 모델에서의 최소 스케쥴을 구하기 위한 유전알고리즘을 제안하였다. 이 알고리즘은 최소 스케쥴을 찾는 동안 사용 가능한 프로세서 수를 넘지 않는 범위 내에서 최적 프로세서의 개수도 구하여 준다. 또한, 기존의 발견적 기법들을 병렬계산 모델에 적용할 때, 주어진 모델이 Coarse Grain(CG)인지 혹은 Fine Grain(FG)인지에 따라 다른 수행도를 보였지만 이 논문에서 소개한 유전알고리즘은 차이를 보이지 않는다. 제안된 유전알고리즘은 유방향무환그래프(Directed Acyclic Graph)의 구조를 갖는 다른 문제에도 적용할 수 있을 것으로 기대된다.

소개한 유전알고리즘은 실행 가능성을 유지하기 위한 별도의 계산이 필요하므로 가능성을 만족하는 새로운 부호화 방법과 교배 연산자 및



[그림 7] C-1 병렬계산모델에서 유전알고리즘의 수행

돌연변이 연산자의 고안이 필요하다. 또한, 본 논문에서 심도있게 다루지 않은 선택 및 복제 방법이나 각 파라미터의 값에 대한 연구도 이루어져야 할 것이다. 지금까지는 다른 기법들과 비교 평가를 수행할 만한 DAG모델들이 소개되어져 있지 않았으므로 본 논문에서는 별도의 DAG 생성기를 코딩하여 대상 모델을 만들었다. 따라서 기준 모델과 평가 기준을 설정하여 기존의 발견적 기법들과 비교 연구할 필요가 있다. 이 연구에서는 다른 대부분의 연구에서와 마찬가지로 모든 프로세서가 동일한 성능을 갖고 있으며 다른 모든 프로세서와 상호 연결되어 있다는 가정을 사용하였다. 그러므로, 실제 시스템에 적용을 위한 연구도 진행되어야 할 것이다.

참 고 문 헌

[1] 김의창, 홍영식, "다중처리기 시스템에서의 태스크할당을 위한 유전알고리즘", 「정보과학회 논문지」, Vol.20, No.1, January 1993, pp.43-51.

[2] 류재철, "N-큐브 다중프로세서 시스템에서 태스크 할당 기법", 「정보과학회 논문지」, Vol.20, No.5, May(1993), pp.739-750.

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

[4] 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, Aug. 1988, pp.1-7.

[5] Charles J. Malmborg, "A genetic algorithm for service level based vehicle scheduling", *Eur. J. Oper. Res.* 1996, pp.121-134.

[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** 1989, pp.225-230.

[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] Colin R. Reeves, "A genetic algorithm for flowshop sequencing", *Computers Ops. Res.*, Vol.22, No.1, 1995, pp.5-13.

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

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

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

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

[14] Gerasoulis A. and Yang T., "On the gra-

nularity and clustering of directed acyclic task graphs", *IEEE Trans. Parallel Distrib. Syst.*, **4** 1993, pp.686-701.

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

[16] Guoyong Shi, "A genetic algorithm applie dto a classic job-shop scheduling problem ", *International Jour. of Syst. Sci.*, Vol.28, No.1, 1997, pp.25-32.

[17] Holland J.H., *Adaption in Natural and Artificial System*, The University of Michigan Press Ann Arbor, 1975.

[18] Hoogeveen J.A., Van de Velde S.L. and Veltman B., "Complexity of scheduling multiprocessor tasks with prespecified processor allocations", *CWI Reprot BS-R9211*, June 1992, Amsterdam.

[19] James C. Bean, "Genetic Algorithms and Random Keys for Sequencing and Optimization," *ORSA Journal on Computing*, Vol. 6, No.2, 1994, pp.154-160.

[20] J.E. Beasley, P.C. Chu., "A genetic algori -thm for the set covering problem", *Euro. J. Oper. Reach.*, 1996, pp.392-404.

[21] Kim S.J. and Browne J.C., "A general approach to mapping of parallel computation upon multiprocessor architectures", *In Proceedings of International Conference on Parallel Processing*, Vol.3, 1988, pp.1-8.

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

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

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

[25] P.C. Chu and J.E. Beasley, "A Genetic Algorithm for the generalised assignment problem", *Comput. Ops. Res.*, Vol.24, No.1, 1977, pp.17-23.

[26] R. Reiter, "Scheduling parallel computation", Journal of ACM, Oct 1968, pp.590-599.

[27] Yang T. and Gerasoulis A., "A fast static scheduling algorithm for DAGs on an unbounded number of processors", *In Proceedings of Supercomputing'91, IEEE*, New York, pp.633-642.

[28] Yang T. and Gerasoulis A., "DSC: Sche -duling parallel tasks on an unbounded number of processors", *IEEE Trans. on Parallel and Distributed System*, Vol.5, No.9, 1994, pp.951-967.

[29] Yang T. and Gerasoulis A., "List scheduling with and without communication delay", *Parallel Comput.*, Vol.19, 1993, pp. 1321-1344.

[30] Yang T. and Gerasoulis A., "PYRROS: static task scheduling and code generation for message-passing multiprocessors", *In proceedings of 6th ACM International Conference on Supercomputing*, Washington DC. 1992, pp.428-437.

[31] Zbiqniew Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1995.