유전 알고리즘을 이용한 다중프로세서 시스템에서의 선형 스케쥴링 알고리즘 구현

(An Implementation of the Linear Scheduling Algorithm in Multiprocessor Systems using Genetic Algorithms)

배 성 환 † 최 상 방 ††

(Sung Hwan Bae) (Sang Bang Choi)

요 약 본 논문에서는 유전 알고리즘을 이용하여 다중프로세서 시스템을 위한 선형 스케쥴링 알고리즘을 제안하였다. 일반적으로 유전 알고리즘은 초기세대를 임의로 생성하기 때문에, 문제에 적합하지 않은 개체들의 영향으로 긴 천이시간과 느린 수렴속도를 갖는다. 제안된 알고리즘은 프로세서간의 통신비용을 고려하며, 초기세대를 생성할 때 현재 노드를 기준으로 직접 후임노드만을 동일 프로세서에 할당함으로써 선형 스케쥴링을 하게 되고, 교배연산과 변이연산에서도 기준 노드의 직접 전임노드나 직접 후임노드의 결합을 변화시킴으로써 선형성을 유지하게 된다. 선형 스케쥴링은 비선형에 비해 프로그램의 병렬성을 최대한 활용할 수 있을 뿐 아니라, 코오스 그레인(coarse grain) 방향성 비순환 그래프(directed acyclic graph: DAG)에서 항상 우수한 스케쥴링 결과를 생성한다. 본 논문의 목적은 유전 알고리즘의 실시간 사용 가능성에 중점을 두었으며, 시뮬레이션 결과 제안된 알고리즘은 대부분의 DAG에서 50세대 내의 빠른 수렴속도를 나타내었다.

Abstract In this paper, we present a linear scheduling method for homogeneous multiprocessor systems using genetic algorithms. In general, genetic algorithms randomly generate initial strings, which leads to long operation time and slow convergence due to an inappropriate initialization. The proposed algorithm considers communication costs among processors and generates initial strings such that successive nodes are grouped into the same cluster. In the crossover and mutation operations, the algorithm maintains linearity in scheduling by associating a node with its immediate successor or predecessor. Linear scheduling can fully utilize the inherent parallelism of a given program and has been proven to be superior to nonlinear scheduling on a coarse grain DAG (directed acyclic graph). This paper emphasizes the usability of the genetic algorithm for real-time applications. Simulation results show that the proposed algorithm rapidly converges within 50 generations in most DAGs.

1. 서 론

고성능 다중프로세서 시스템에서 효율적인 프로그램 처리를 위한 소프트웨어 기술로는 컴파일러 기법과 병렬성이 높은 프로그램 개발 등이 있다. 그 중 컴파일러 기술은 분할된 모듈들을 프로세서에 할당하여 선행관계(precedence relation)[1]를 유지하면서 전체 작업 완료시간(task completion time)을 최소화하는 스케쥴링 기법과 작업 부하를 균일(load balancing)하게 분배하는 방법과 밀접한 관계를 가지고 있다[2]. 스케쥴링은 논리적으로 분할된 프로그램 모듈간의 작업 의존상태를 나타내는 방향성 비순환 그래프 (directed acyclic graph: DAG)에서 작업노드의 집단화(clustering)를 이용하는 방법이 많이 연구되고 있다. 집단화는 DAG상의 노드를 여러개의 집단으로 사상하여 각 프로세서에 할당하는 것이다. DAG의 집단화는 비선형적인 방법과 선형적인 방법으로 나눌 수 있으며, 비선형 집단화는 같은 집단에

· 본 연구는 1999년도 정보 통신부 '정보 통신 우수 대학원' 지원 사업의 일환으로 수행하였음.

† 비 회 원 : 인하대학교 전자공학과
mago51@hotmail.com
†† 정 회 원 : 인하대학교 전자공학과 교수
sangbang@dragon.inha.ac.kr

논문접수 : 1998년 9월 3일
심사완료 : 1999년 11월 11일

독립적인 작업들이 포함되는 경우이고 그렇지 않은 경우는 선형 집단화이다. 이러한 집단화의 대상이 되는 DAG는 결정성(granularity)을 사용하여 특징지울 수 있다. 결정성이란 작업노드의 통신비용에 대한 실행비용의 비율이며, 일반적으로 1보다 크거나 같은 경우를 코오스 그레인(coarse grain)이라 하고 1보다 작은 경우를 파인 그레인(fine grain)이라 한다[3,4].

다중프로세서 스케쥴링 문제를 해결하기 위해 다양한 접근 방법들이 제시되고 있다. 경험적 방법으로는 Kashara와 Narita[5,6]가 제시한 CP(Critical Path) 알고리즘과 DFS(Depth First Search) 알고리즘이 있다. Chen[7]의 경우 SSS(State-Space Search) 알고리즘을 제안하였으며, Hellstrom과 Kanal[8]은 신경망 모델을 이용하여 다중프로세서 스케쥴링 문제를 해결하기 위해 연구하였다. 본 논문에서는 기존의 경험적 방법에서 나타나는 작업 그래프의 형태와 관련된 유연성 결여 문제를 해결하기 위해 최근 NP-complete으로 알려진 여러 문제의 해결 방법으로 제시되고 있는 유전 알고리즘을 이용한 다중프로세서 스케쥴링 방법을 제안한다.

유전 알고리즘은 다양한 스케쥴링 알고리즘에 적용되어 그 사용가능성에 대한 연구가 진행되어왔다. Ono와 Tsugawa[9]는 크루(crew) 스케쥴링 문제를 풀기 위하여 유전 알고리즘을 적용하였으며, Jozefowska, Rozycki[10] 등은 불연속-연속(discrete-continuous) 스케쥴링 문제에 적용하여 보았고, Ramat, Venturini [11] 등은 자원제한 프로젝트 (resource-constraint project) 스케쥴링 문제에 시도하여 보았다. Hou, Ansari [12,13] 등은 유전 알고리즘에서 개체를 표현하기 위한 스트링(string)을 사용하여 다중프로세서 스케쥴링에 적용하였으며, Wang과 Korfhage[14]는 동일한 기법을 사용하였으나 다만 개체를 표현하기 위하여 2차원 매트릭스 인코딩(matrix encoding)을 사용하였다. Correa Ferreira[15] 등은 스케쥴링 정보를 유전 알고리즘과 혼합하여 다중프로세서 스케쥴링을 시도하였으며, Schwehm과 Walter[16]는 트랜스퓨터(transputer)로 구성된 다중프로세서를 위한 스케쥴링을 유전 알고리즘을 이용하여 어레이 프로세서에서 구현하였다.

본 논문은 다중프로세서 스케쥴링에서 유전 알고리즘의 실시간 사용 가능성에 중점을 두었다. 유전 알고리즘을 실제 응용 프로그램에 적용하기 위해서는 주어진 프로그램의 스케쥴링으로는 적합하지 않은 일부 초기 세대에 의해 과도기적으로 나타나는 매우 느린 수렴 문제를 해결해야 한다. 본 논문에서는 선형 집단화를 이용한 스케쥴링으로 다른 전처리 단계를 거치지 않고 수렴속도를 향상시키고 있다. 이러한 선형 스케쥴링은 프로그램의 병렬성을 최대한 활용할 수 있을 뿐아니라, 각 노드의 실행비용이나 통신비용의 변화에 비교적 민감하지 않으며, 코오스 그레인 DAG에서는 비선형 집단화보다 항상 우수한 스케쥴링 결과를 생성할 수 있다[3]. 시뮬레이션에서는 일부 노드들의 실행비용을 변화시켜 발생되는 파인 그레인 DAG에 대하여 유전 알고리즘을 사용한 선형 스케쥴링 방법이 효과적인지에 대한 성능평가도 이루어진다. 기존의 연구 중 본 논문과 비교 가능한 알고리즘인 Hou의 유전 알고리즘[12,13]에서는 최적화될 수 있는 특정한 해가 교배나 변이 과정을 통해 발생될 가능성이 전혀 없을 수도 있는 문제가 있다. 본 논문에서는 이러한 문제점을 보안한 새로운 유전자 표현법과 유전연산 및 스케쥴링 방법을 제시한다.

시뮬레이션을 통하여 얻은 결과를 분석해 보면 임의로 생성한 대부분의 작업 그래프에서 50세대 내외의 빠른 수렴을 보였다. Hou의 알고리즘보다 평균적으로 2.09배 빨리 수렴하였으며 최종해에 대하여도 5.06%의 작업 완료 시간이 감소하였다. 또한, 각각 10%, 20%, 30%에 해당하는 노드의 작업 실행비용을 임의로 변경한 후 이전의 스케쥴링 정보를 이용하여 재스케쥴링하는 경우, 제안된 알고리즘은 Hou의 알고리즘과 비교하여 각각 2.55배, 2.6배, 2.52배 빨리 수렴하며, 작업 완료 시간도 3.41%, 3.19%, 3.22% 향상됨을 알 수 있다. 각 세대에서 가장 우수한 개체를 선택하는 엘리트 보존 정책을 사용하였기 때문에 두 방법의 작업 수행 시간간의 차이는 크지 않다.

본 논문의 구성은 다음과 같다. 제 2장에서는 다중프로세서 시스템에서 DAG의 집단화로 표현되는 스케쥴링 문제를 설명하고, 일반적인 유전 알고리즘의 이론과 이를 스케쥴링에 적용하는 방법에 대해 기술한다. 제 3장에서는 제안된 알고리즘에 대한 유전자 표현법, 적합도 함수에 따른 개체 선택방법 및 유전 연산자에 대하여 설명한다. 제 4장에서는 제안된 알고리즘에 대한 검증 및 기존의 알고리즘과 비교하기 위하여 수행한 시뮬레이션 결과에 대한 분석을 기술하며, 끝으로 제 5장에서는 본 연구에 대한 결론을 맺는다.

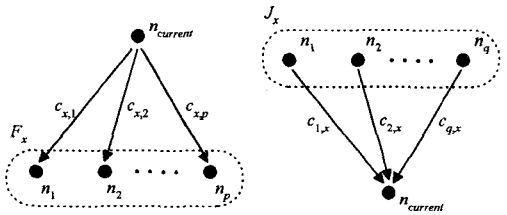
2. 유전 알고리즘을 이용한 스케쥴링

2.1 다중프로세서 스케쥴링

본 논문의 스케쥴링에서 사용되는 DAG는 $G = \{V, E, C, T\}$로 정의한다. V는 DAG를 구성하는 모든 작업 노드의 집합을 나타내고 $|V|$는 그 개수이다. V를 구성하는 각 노드는 $V = \{n_i, i = 1, 2, ..., |V|\}$로 나타낸다.

E는 노드와 노드간의 모든 통신 링크의 집합을 나타내며, $|E|$는 통신링크의 전체 개수이다. 노드 n_i와 n_j를 연결하는 방향성 통신 링크는 $e_{i,j} = (n_i, n_j)$로 나타낸다. C는 링크를 실수의 통신비용으로 사상하는 함수이며, T는 노드를 작업 실행비용으로 사상하는 함수이다. 즉, 함수 C는 $C : E \rightarrow R$이며, 여기서 R은 실수의 통신비용을 나타내고 $C(e_{i,j})$는 링크 $e_{i,j}$의 통신비용 $c_{i,j}$를 의미한다. 함수 T는 $T : V \rightarrow R$이며, 여기서 R은 실수의 작업비용, $T(n_i)$는 노드 n_i의 작업 실행비용 t_i를 의미한다.

DAG를 구성하는 작업노드들은 일반적으로 그림 1과 같은 두 종류의 형태로 분류할 수 있다. 본 논문에서는 그림 (a)와 같은 형태를 분기노드(fork node), (b)와 같은 형태를 결합노드(join node)라고 한다. 현재 작업노드를 기준으로 거리(distance)가 1인 노드들은 다시 직접 전임노드(immediate predecessor) 또는 직접 후임노드(immediate successor)로 분류할 수 있다. 직접 후임노드는 현재 작업노드에 종속된 노드들 중에서 거리가 1에 해당하는 노드들이며, 직접 전임노드는 현재 작업노드가 종속되어진 노드들 중에서 거리가 1에 해당하는 노드들이다. 그림 (a)의 $n_{current}$에 대해 n_1, n_2, ..., n_p은 직접 후임노드들이며, (b)의 $n_{current}$에 대해 n_1, n_2, ..., n_q은 직접 전임노드들이다.



(a) 분기노드 (b) 결합노드

그림 1 작업노드의 분류

DAG의 각 노드는 프로그램을 논리적으로 분할한 모듈을 의미하고, 이 노드들을 작업의 실행 순서를 유지하면서 한 집합으로 묶는 것을 집단화라고 한다. 즉, 집단은 작업노드들의 부분집합이고 이 집합에 포함된 작업노드들은 동일한 프로세서에서 실행되므로 같은 집단에 포함된 작업노드들 간의 통신비용은 무시될 수 있다. 이러한 집단화를 거친 후 각 집단을 프로세서에 할당하고 각 작업의 실행 시작 시간을 결정하는 것을 스케쥴링이라 한다. 다중프로세서 시스템에서 주어진 프로그램에 대한 스케쥴링은 DAG로 표현되는 여러 가지 형태의 작업 그래프(task graph)를 이용하여 프로그램의 전체 수행시간이 최소화되도록 프로그램 모듈을 프로세서에 할당하여야 하며, 필요한 모듈들이 모두 수행되어야 한다. 그러나 프로그램을 병렬로 처리하면 데이터간의 종속성(dependency) 문제가 발생하기 때문에 작업 그래프에 의하여 정해진 선행관계를 유지하면서 각 모듈을 처리해야 한다. 따라서 위에서 설명한 데이터간의 선행관계를 유지하면서 작업 부하를 균일화하는 문제는 다중프로세서 시스템을 위한 스케쥴링에서 고려해야 할 가장 중요한 요소이다.

단일 프로세서 시스템과 달리 다중프로세서 시스템에서는 각 프로세서에서 수행되는 작업들의 수행시간 외에도 프로세서간의 통신비용(communication cost)이 주는 영향이 커지게 된다. 특히 파인 그레인의 경우 프로세서에서 데이터 자체를 처리하는 시간보다 프로세서간에 그 결과를 주고받는데 더 많은 시간이 소요됨으로 공간적으로 흩어진 프로세서에 작업을 할당함으로써 발생하는 통신비용은 다중프로세서 시스템의 전체 성능을 결정하는 가장 중요한 요소 중의 하나이다. 그러나 다중프로세서 시스템에서 최적화된 스케쥴링을 찾는 것은 이미 NP-complete 문제[17]로 알려져 있기 때문에 시스템의 위상(topology)에 따른 통신비용을 고려하여 최적에 가까운 해(near-optimal solution)를 구하기 위한 여러 가지 경험적 방법들이 제안되었다.

Gerasoulis와 Yang[3]은 코오스 그레인으로 판명된 임의의 그래프에 대해 비선형 집단화보다 같거나 우수한 선형 집단화가 항상 존재함을 증명하였으며, 코오스 그레인 DAG에 대한 임의의 비선형 집단화 결과는 프로그램 완료시간이 같거나 짧은 선형 집단화로 변형시킬 수 있음을 보였다.

또한 그들은 정의한 결정성을 사용하여 $PT_{opt} \leq PT_{lc} \leq (1 + 1/g(G))PT_{opt}$임을 증명하였다. 여기서 PT_{opt}는 최적으로 스케쥴링된 경우의 프로그램 완료시간이며, PT_{lc}는 선형 집단화에 의해 스케쥴링된 것의 프로그램 완료시간이고, $g(G)$는 주어진 방향성 비순환 그래프 G의 결정성이다. 특히, 주어진 DAG가 코오스 그레인인 경우 $g(G) \geq 1$이므로 $PT_{lc} \leq 2 \cdot PT_{opt}$임을 알 수 있다.

Hou와 Ansari[12,13]는 유전 알고리즘을 이용한 스케쥴링 방법을 제안하였으나 작업 그래프를 구성하는 중요한 요소 중에 하나인 통신비용을 전혀 고려하지 않아 실제 프로그램 적용에 한계가 있으며, 제한된 수의 프로세서를 이용한 스케쥴링 방법에 대하여만 연구하였으며, 유전연산을 적용할 경우 교배와 변이연산이 자유

롭지 못한 제약 조건이 발생한다. 또한 일반적인 유전 알고리즘에서와 같이 초기세대에서 무작위로 개체를 생성하기 때문에 실제적으로는 의미 없는 개체가 일부 생성되어 결과적으로 최종해로의 수렴을 느리게 한다. 따라서 본 논문에서는 개체들 사이에 교배와 변이연산이 자유롭고, 우수한 초기세대를 만드는데 효과적인 선형 집단화를 이용하여 주어진 작업 그래프의 변화에 적응력이 뛰어난 유전 알고리즘을 제안한다.

2.2 유전 알고리즘

유전 알고리즘은 임의의 함수 $y = G(x)$의 최적해를 발견하는 모의 진화(Simulation Evolution)형 탐색 알고리즘의 성격을 가지고 있으며, John Holland는 알고리즘의 진화 과정을 어려운 문제 해결의 기법으로써 컴퓨터 알고리즘과 결합시켜 유전 알고리즘을 탄생시켰다[18,19]. 이미 유전 알고리즘은 패턴 인식이나 기계 학습, 로봇 공학, 순회 판매원 문제(TSP: Traveling Salesman Problem)를 비롯한 최적화 문제와 다양한 응용분야에 적용되고 있다[13].

일반적인 유전 알고리즘은 그림 2에서와 같은 처리 순서를 통해 진화된다. 첫번째 단계는 초기화 단계이다. 초기화 단계에서는 탐색공간에서 무작위로 초기 집단을 생성한다. 두번째 단계에서는 초기화 단계에서 무작위로 생성된 각 개체의 적응 정도를 적합도 함수에 따라 계산한다. 세번째 단계에서는 선택전략에 따라 다음 세대로 전승될 유전자를 선택한다. 네번째 단계에서는 적합도에 따라 개체를 선택한 후, 선택된 개체간의 교배와 변이연산을 통해 새로운 개체를 생성한다. 마지막 단계에서는 수렴조건에 해당될 때까지 두번째 단계에서 네번째 단계까지를 반복 수행한다.

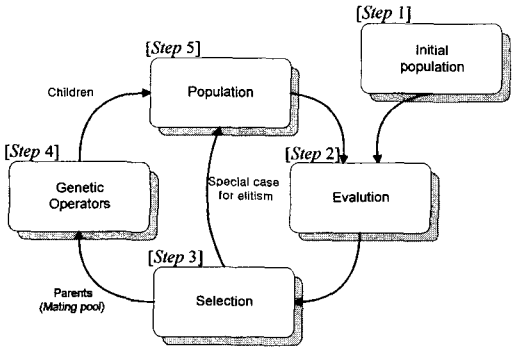


그림 2 유전 알고리즘의 진행 사이클

2.3 유전자형의 표현법과 초기세대

유전 알고리즘의 입출력으로 사용되는 자료를 '염색체(chromosome)' 또는 '문자열(string)'이라고 한다. 현재까지 대부분의 염색체는 이진 문자열을 이용하여 표현해왔는데, 최근 들어서는 문제의 성격과 특징에 따라 문자열을 정수 또는 실수로 사용하기도 한다. 문제에 따른 최적의 해를 탐색하기 위해서는 그 문제가 가지고 있는 정보를 정확하고 효과적으로 표현하여야 한다.

스케줄링 문제를 해결하기 위한 유전 알고리즘에서 염색체를 표현할 때 이진 문자열을 사용하는 것보다 실제 스케줄에 가까운 정수를 사용하는 것이 문제의 특성에 더 적합하다고 알려져 있다. 정수 문자열을 사용하면, 이진 문자열을 사용할 때 거쳐야 하는 부호화 과정과 복호화 과정이 필요없고, 연산 후에 부적당한 염색체가 생성되는 것을 방지할 수 있다. 그림 3은 문제 공간에서 생성되는 인자형과 실제 해공간에서 나타나는 표현형에 관한 개념도이다. 본 논문에서는 염색체를 표현하기 위해서 스케줄 자체를 나타내는 작업노드의 번호를 정수 문자열로 사용한다.

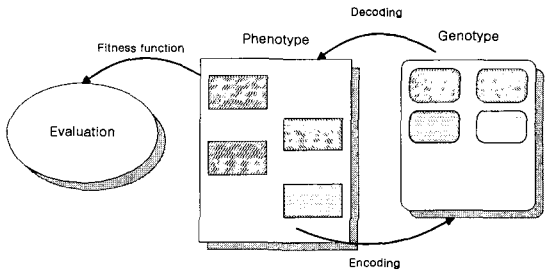


그림 3 인자형과 표현형의 변환 관계

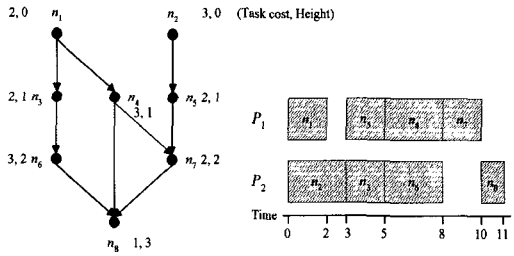


그림 4 작업 그래프와 2개의 프로세서를 사용한 스케줄의 예

그림 4는 Hou의 알고리즘에서 사용한 스케줄링의 표현법이다. 노드는 각각 작업 실행비용과 높이(Height) 정보를 가지고 있으며, 유전자를 표현하기 위해 작업노

드의 번호를 이용하여 정수형 문자열로 나타내었다. 이때 정수형 문자열의 순서는 프로세서에서 각각의 작업 노드들은 선행관계를 나타내고 있다. 그림의 예에서와 같이 스케쥴링에서 프로세서간의 통신비용은 고려되지 않았다.

2.4 적합도 함수와 개체 선택

적합도 함수(fitness function)는 유전자 집합에서 각각의 유전자가 문제의 환경에 얼마나 잘 적응하는가를 나타내는 척도로서 개체간의 적합도에 관한 측정과 평가를 위해서는 다음 조건을 만족하여야 한다. 먼저 궁극적으로 적합도 함수가 어떤 형태로든 표현이 가능해야 하며, 또한 수치적으로도 계산이 가능하여야 한다. Hou의 알고리즘에서는 스케쥴링의 적합도 함수를 다음과 같이 정의하였다.

$ftt(i)$ = Finishing time of task n_i
$ftp(j)$ = Finishing time for processor P_j

따라서, 각 스케쥴링의 최종 프로그램 완료시간은 다음 식과 같다.

$$FT = Max\ \{ftp(j) \text{ for each processor } P_j\}$$

적합도 함수는 세대의 변화에 따른 증가 함수를 표현하는 것이 일반적이므로 아래와 같은 식으로 표현된다.

$$Fitness\ function = \frac{1}{FT}$$

적합도 함수에 따라 적응도가 정해진 개체들은 평가치에 따른 자연선택(natural selection)이 일어나게 된다. 각 세대에서 적합도 함수에 따라 선택이 이루어지게 되며, 개체를 선택하는 다양한 방법들이 제시되었다. 대표적인 선택 방법으로는 룰렛 휠(roulette wheel)방식과, 토너먼트 방식, 그리고 엘리트 전략이 있다. 이러한 선택에 의해 재구성된 다음 세대에서는 유전 알고리즘의 본질적인 탐색 기능에 해당하는 교배와 변이 과정을 거쳐서 새로운 형질을 가진 개체가 생성된다.

2.5 유전 연산자

유전연산은 이전의 세대에서 발생된 개체들 중에서 적합도가 뛰어난 우성 개체들의 성질을 다음 세대로 전달하는 역할을 한다. 새로운 염색체를 생성하기 위하여 교배연산과 변이연산을 수행한다. 이때 생성된 염색체들에 적당한 해(legal solution)와 부적당한 해(illegal solution)가 모두 존재할 경우, 각 개체에 대해 정해진 규칙에 위배되는지의 여부를 통한 부적당한 해의 제거 작업에서 소요되는 시간이 수렴속도에 큰 영향을 끼치게 되므로 생성되는 염색체가 항상 적당한 해를 나타내도록 하는 것이 중요하다.

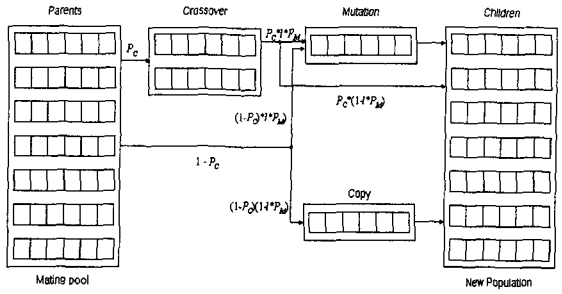


그림 5 유전 연산자를 사용한 유전자 생성 과정

그림 5는 부모 유전자에 유전 연산자를 적용하여 새로운 개체를 생성하는 과정을 나타낸 것이다. 지정된 개체수(population size)로 구성된 유전자 풀(mating pool)에서 임의로 유전자를 선택하여 유전연산을 통해 새로운 개체를 만들어 다음 세대를 구성한다. 이때 교배연산을 수행할 개체들은 교배확률 P_C를 통해 유전자 풀에서 2개의 부모 개체를 선택한다. 나머지 연산들은 확률 $(1 - P_C)$을 통해 부모 개체를 선택한다. 그리고, 변이연산에 이용될 개체들은 변이확률 P_M을 통해 선택되어진다. 이때 개체가 l개의 염색체로 구성되어 있다면 개체가 변이될 확률은 다음 식과 같다.

$$P_{Mutation} = l \cdot P_M$$

따라서 교배연산을 수행하지 않고 변이연산을 수행할 확률은 아래 식과 같다.

$$P_{No\ cross,\ Mutation} = (1-P_C) \cdot l \cdot P_M$$

교배연산을 수행한 후 다시 변이연산을 수행할 확률은 다음과 같이 얻어진다.

$$P_{Cross,\ Mutation} = P_C \cdot l \cdot P_M$$

교배연산은 수행하나 변이연산은 수행하지 않을 확률은 다음과 같다.

$$P_{Cross,\ No\ mutation} = P_C \cdot (1-l \cdot P_M)$$

유전자 풀로부터 선택되어진 부모 개체 중에서 교배연산이나 변이연산을 전혀 거치지 않고 개체를 그대로 복사하게 될 확률은 다음 식과 같이 주어진다.

$$P_{Copy} = (1-P_C) \cdot (1-l \cdot P_M)$$

교배(crossover)는 두 개체 사이의 염색체 교환을 통해서 새로운 개체를 생성하는 것으로 양쪽 부모의 우수한 부분형질을 적절히 조합하여 자식에게 전승시키는 것에 성공한다면 우수한 개체생성 및 수렴속도에서 비약적인 발전을 가져오게 된다. 일반적인 단순 교배 방법에서는 임의의 한 교배점(crossover site)을 정하고 교배점을 중심으로 부모의 부분형질을 조합하게 된다. 본 논문의 경우 교배점으로 무작위 레벨이 선택되고 정해진 레벨을 중심으로 서로의 부분형질을 교환하게 된다. Hou가 제안한 방법[12,13]에서는 교배점의 위치를 제한하게 되므로 특정한 유전자형의 스케쥴이 생성되지 않을 수도 있다. 특정한 유전자형을 통해서만 최적화된 해가 생성될 수 있다면 교배점의 위치를 제한하는 방식의 교배 방법에서는 최적화된 해를 만들 수 있는 이러한 후보 유전자가 생성되지 않을 수도 있다는 것을 의미한다.

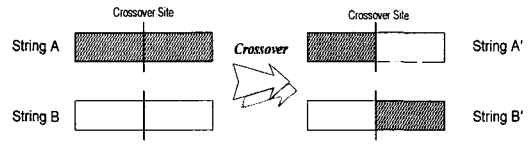


그림 6 교배연산의 예

또 다른 유전연산인 변이(mutation) 작용은 염색체상의 어떤 유전자 위치의 값을 다른 대립 유전자와 바꾸어 넣어 새로운 개체를 생성하는 국소적인 무작위 탐색(random search)의 한 종류이다. 변이는 실행 불가능하거나 적합하지 않은 해를 나타내는 염색체, 즉 치사 유전자를 만들어낼 위험성을 내포하고 있는 반면, 집단으로서 상실한 대립 유전자의 회복에 기여하는 경우도 있어서 집단의 다양성을 유지하기에 유용한 작용의 하나라고 볼 수 있다. 본 논문에서는 무작위로 정해진 노드를 변이 유전자 위치로 정하고 해당되는 결합노드나 분기노드의 결합을 변화시켜 선형성을 보장하는 다양한 스케쥴링을 생성하게 된다.

그림 7 변이연산의 예

3. 제안된 스케쥴링 알고리즘

본 논문에서 제안하는 스케쥴링 방법은 선형 집단화에 바탕을 둔 유전 알고리즘으로 다음과 같은 가정을 사용한다.

첫째, 각 프로세서에 할당된 작업의 수행은 비선취형(nonpreemptive)으로 이루어지는 것을 가정한다. 즉, 작업의 수행에 필요한 데이터가 모두 준비되면 스케쥴링된 시점에서 작업을 시작한다. 프로세서에 할당된 모듈은 일단 수행이 되면 도중에 멈출 수 없으며, 작업이 끝나면 직접 후임노드에 해당하는 모듈들이 할당된 모든 프로세서에게 수행결과를 전송한다. 수행결과를 다른 프로세서들에게 전송하는 통신은 병렬로 이루어진다.

둘째, 같은 집단에 포함된 작업노드들은 한 프로세서에 할당되므로, 이들 작업노드들 사이의 통신비용은 서로 다른 집단에 포함된 작업노드들 사이의 통신비용에 비해 무시할 수 있다고 가정한다. 즉, 종속적인 두 작업노드를 같은 집단에 포함시키면 이 작업노드들 사이의 통신비용은 무시된다.

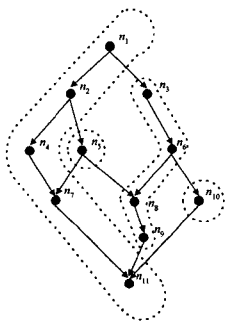
셋째, 스케쥴링된 프로그램은 완전 연결된 다중프로세서 구조에서 수행되는 것으로 가정한다. 여기서 각 프로세서는 동일한 성능을 가지며 무한히 많이 존재할 수 있다. 이 조건은 제안된 알고리즘이 어떤 특정한 다중프로세서 구조에 제한된 스케쥴링보다는 프로그램 수행시간을 최소화하기 위한 일반적인 스케쥴링 알고리즘 자체에 중점을 두고 있음을 의미한다.

3.1 유전자형 표현법 및 초기세대

본 논문에서는 스케쥴링 문제를 유전 알고리즘으로 해결하기 위해 해당 스케쥴을 그림 8에서와 같이 각 프로세서에 할당된 작업노드의 번호를 이용한 정수형 문자열로 나타낸다. 이는 각 프로세서에 어떤 작업노드가 할당되느냐에 따라 스케쥴링이 결정되기 때문이며, 작업간의 선행관계는 레벨(level) 정보를 이용하여 결정한다. 그림 8은 작업 그래프의 예와 본 논문에서 사용되는 유전자 표현인 정수형 스트링을 도식화한 것이다. 각 유전자는 여러개의 선형집단으로 구성되어 있고, 각 집단에는 할당된 작업노드의 번호가 정수로 표시되어 있다. 한 프로세서에 할당된 프로그램 모듈들의 실행 순서는 해당 작업노드의 레벨 정보를 통해서 이루어진다.

```
FUNCTION Initial_String(G)
{ while (There is an unscheduled node) {
      Select a node n_current with the highest level among
      unscheduled nodes;
      Assign n_current to a new cluster;
      while (There is an unscheduled successor) {
          if (n_current is a fork node)   // In terms of
              unscheduled nodes
                  n_successor = Select a node randomly from
```

```
                unscheduled immediate successors;
        else  // Single successor
                n_successor = Select the unscheduled
                  successor;
        Assign n_successor to the cluster to which n_current
          belongs;
        n_current = n_successor;
      }
    }
  }
```



	P_1	P_2	P_3	P_4
Level 0	n_1			
Level 1	n_2	n_3		
Level 2	n_4	n_6	n_5	
Level 3	n_7	n_8		n_{10}
Level 4		n_9		
Level 5	n_{11}			

P_1	1	2	4	7		11
P_2		3	6	8	9	
P_3			5			
P_4				10		

그림 8 작업 그래프와 유전자형 표현의 예

초기 세대를 생성하기 위한 위의 알고리즘을 설명하면 다음과 같다. 파라미터 G는 DAG로서 노드의 개수와 작업 실행비용, 통신비용 및 연결 상태로 정의된다. 먼저 루트 노드를 현재 노드로 하여 집단화를 시작한다. 현재 노드 $n_{current}$가 2개 이상의 링크를 가지는 분기 노드일 경우 직접 후임노드 중에서 하나를 임의로 선택하여 동일 집단으로 묶은후, 선택된 직접 후임노드를 현재 노드로 하여 더 이상 선택할 직접 후임노드가 없을 때까지 계속 진행한다. 다시 스케쥴링 되지 않은 최상위 레벨의 노드를 현재 노드로 선정한 후 직접 후임노드만을 선택하는 선형 집단화를 모든 노드가 스케쥴링될 때까지 계속한다. 이러한 방법은 분기노드와 결합노드에서 임의로 집단화를 수행하게 되므로 선형 집단화를 보장하면서 무작위로 다양한 스케쥴을 생성할 수 있다.

3.2 적합도 함수 및 개체 선택

유전 알고리즘에서 개체를 선택하는 절대적 평가 기준인 적합도 함수는 전체 프로그램의 완료시간, 즉 각 프로세서에서 모든 작업의 수행이 끝나는 최종 시간으로 설정하였다. 이것은 스케쥴링의 목적이 전체 프로그램의 완료시간을 최소화시키는데 있으므로 생성된 유전자들의 선택 기준은 당연히 프로그램이 종료될 때까지 걸리는 시간이 된다. 일반적으로 적합도 함수는 세대가 진행될수록 해당되는 값이 커지는 증가 함수(increasing function)로 표현함으로 본 논문에서는 프로그램 완료시간의 역수를 사용한다

$$Fitness\ function = \frac{1}{Max\ \{T_i \mid i = 1, 2, \ldots, p\}}$$

$$T_i = Execution\ completion\ time\ of\ processor\ P_i$$

위의 식에서 T_i는 프로세서 P_i에 할당된 모든 작업이 종료되는 시간을 의미하며, p는 프로그램 수행에 사용된 프로세서의 개수이다. 즉, 프로그램의 종료시간은 각각의 프로세서가 그 수행을 종료하는 시간의 최대값을 선택함으로써 구할 수 있다. 이 최대값의 역수가 제안된 알고리즘의 목적함수가 되며, 적합도에 따라 선택된 개체들은 다음 세대에서 다양한 방법의 교배와 변이를 통한 유전연산을 수행하게 된다.

본 논문의 경우에는 2가지 선택 전략을 병행하여 사용한다. 첫 번째로 토너먼트 선택(tournament selection)을 수행하는데, 이 방법은 집단으로부터 어떤 결정된 수(본 논문의 경우 3개)의 개체를 무작위로 선택하여, 그 중에서 적응도가 가장 높은 개체를 선발하는 절차를 다음 세대로 넘기고 싶은 수만큼의 개체가 구성될 때까지 반복한다. 두 번째로 엘리트 보존 전략(elite preservation strategy)을 사용한다. 이는 집단 중에서 가장 적응도가 높은 개체를 그대로 다음 세대로 넘기는 방법이다. 엘리트 보존 전략은 해당 세대의 엘리트 개체가 교배나 변이연산에 의해 파괴될 가능성이 있으므로 엘리트 개체는 다음 세대로 곧바로 전승한다. 본 논문에서는 토너먼트 선택 후 구성된 다음 세대의 집단에서 적응도가 가장 낮은 개체와 이전 세대에서 적응도가 가장 높은 개체를 교환하는 방식을 사용한다.

3.3 유전 연산자

유전 알고리즘은 교배를 통해 원하는 해로 수렴하는 작용을 하고, 다시 변이를 통해 임의의 영역으로 탐색공간을 넓혀 최적의 해를 찾는다. 본 논문에서는 선형 집단화를 보장하는 교배와 변이 유전 연산자를 제안한다. 다음은 제안된 알고리즘에서 사용되는 교배연산 함수이다.

```
FUNCTION Crossover(String_A, String_B, P_C)
{ If((Random number between 0 and 1) ≤ P_C) {
      Cross = Select a level randomly for the crossover
```

```
    operation;
  While (There is an unprocessed node with Cross
    level) {
      Let n_cross be an unprocessed node with Cross
        level;
      Select two clusters that contains n_cross from each
        of String_A and String_B;
      Construct new clusters by exchanging
        descendants of n_cross in the two clusters;
  }
  If(There are clusters that consists of only those
   nodes with Level > Cross)
      Interchange those clusters between String_A and
        String_B;
 }
}
```

위의 함수에서 파라미터 *String_A*와 *String_B*는 교배연산이 수행될 개체이며, 두 개체에 대하여 교배가 이루어질 확률은 P_C이다. 먼저 교배위치 *Cross*를 임의의 레벨값으로 정한다. 이 레벨값을 갖는 노드를 기준으로 노드간의 집단화 방법을 교환하며, 이때 생성되는 개체는 선형 집단화가 보장되도록 한다. 먼저 해당 레벨의 한 노드를 공동으로 포함하는 두 개의 클러스터를 *String_A*와 *String_B*에서 찾은 후 그 노드의 후임노드들을 서로 교환한다. 선택된 레벨의 모든 노드에 대하여 동일한 동작을 반복 수행한다. 마지막으로 선택된 레벨값보다 큰 노드들로만 구성된 클러스터가 *String_A*나 *String_B*에 있으면 서로 교환한다.

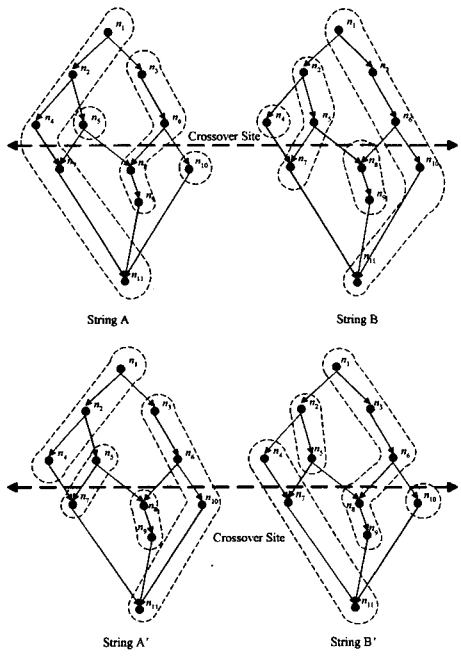


그림 9 교배연산을 통해 생성된 새로운 개체의 예

그림 9는 위의 방법에 따라 String A와 String B에 대하여 수행된 교배연산의 예를 그림으로 표현한 것이다. 교배 위치로 선택된 레벨을 기준으로 새로운 개체 A'의 상위레벨 부분은 이전의 개체 A에서, 하위레벨 부분은 이전의 개체 B로부터 전수받아 결합된다. 동일한 방법으로 새로운 개체 B'의 상위레벨 부분은 이전의 개체 B의 상위레벨 부분을 전수받고, 하위레벨 부분은 A에서 전수받아 생성된다.

제안된 교배 방법에 의해 작업 그래프에서 발생할 수 있는 모든 형태의 선형 스케줄을 생성할 수 있고, 유전 알고리즘의 적합도에 따라 세대가 진행될수록 우수한 스케줄은 살아남아 다음 세대의 생성에 지배적 영향을 끼치게 된다. 따라서, 본 논문에서는 유전 알고리즘의 코드화 평가규범 가운데 문제공간 상에서 '해' 후보를 모두 염색체로 생성할 수 있는 완전성(completeness)을 보장받게 된다.

```
FUNCTION Mutation(String_M, P_M)
{ If((Random number between 0 and 1) ≤ P_M) {
    Select a node n_select randomly in String_M;
    if(n_select is a fork node)
      n_mutation = Select a node from immediate
        successors contained in different clusters;
    else if(n_select is a join node)
      n_mutation = Select a node from immediate
        predecessors contained in different clusters;
    else
      n_mutation = Select any node from successors or
        predecessors contained in different clusters;
    Mutate String_M in a way that n_select and n_mutation
      are contained in the same cluster;
  }
}
```

위의 함수에서는 선형 집단화를 유지하는 변이연산이 수행되며, 파라미터는 변이연산이 적용될 개체 *String_M*과 동작이 일어날 확률 P_M이다. 먼저 주어진 개체에서 임의의 노드 n_{select}을 선택한 후, 그 노드를 기준으로 변이를 수행한다. 이때 n_{select}가 분기노드인 경우 직접 후임노드 중에서 n_{select}와 다른 집단에 속한 노드 하나를 임의로 선택한다. 이때 선택된 $n_{mutation}$의 하부구조(집단화된 형태)를 노드 n_{select}가 흡수한다. 이와 달리 결합노드인 경우 직접 전임노드 중에서 n_{select}와 다른 집단에 속한 노드 하나를 $n_{mutation}$으로 선택하여 변이를 수행한다. 이때 선택되어진 $n_{mutation}$의 상부 구조를 노드 n_{select}

가 홉수한다. 변이를 수행하는 노드가 단일 링크로 연결된 노드인 경우는 n_{select}와 같은 집단으로 묶이지 않은 전임노드나 후임노드 중에서 임의로 한 노드를 $n_{mutation}$으로 선택한다. 선택된 두 노드 n_{select}와 $n_{mutation}$은 같은 집단에 포함되도록 변이연산을 수행한다.

그림 10은 위의 방법에 따라 주어진 개체에 대하여 변이연산을 수행한 결과를 그림으로 표현한 것이다. 임의로 선택된 노드 n_8을 기준으로 변이연산을 수행한다고 가정한다. n_8은 결합노드에 해당하며 직접 전임노드는 n_5와 n_6이다. 이 가운데 n_5를 선택할 경우 그림 10의 왼쪽에 나타난 바와 같이 n_5, n_8, n_9을 동일한 프로세서에 할당하는 스케쥴을 생성하게 된다. 반면, n_6를 선택할 경우 그림의 오른쪽에 나타난 것과 같이 n_3, n_6, n_8, n_9을 동일한 프로세서에 할당하는 스케쥴을 생성하게 된다. 주어진 예에서는 n_8을 기준으로한 변이연산을 통하여 두 가지의 새로운 개체를 생성할 수 있다. 제안된 변이연산 방법은 결합노드가 아닌 분기노드에서도 직접 후임노드와의 결합을 변화시킴으로써 선형성을 유지하면서 연산 전의 개체 특성을 포함하는 다양한 개체들을 생성할 수 있다.

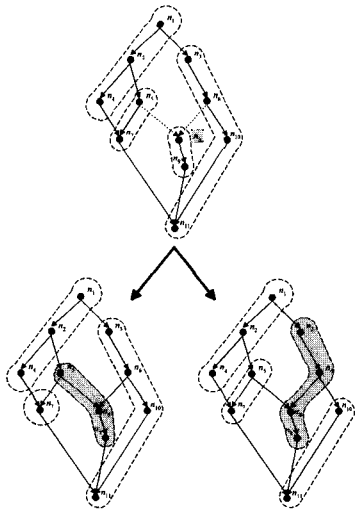


그림 10 변이연산을 통해 생성된 새로운 개체의 예

그림 11과 같이 4개의 노드로 구성된 다이아몬드형 작업 그래프를 통해 제안된 알고리즘과 Hou의 알고리즘의 완전성을 비교할 수 있다. 다이아몬드형 작업 그래프에서 선형 집단화를 사용한 스케쥴링 방법은 그림에서와 같이 모두 4가지가 가능하다. 그림에서 (a)와 (b)의 경우는 제안된 알고리즘이나 Hou의 알고리즘에서 초기세대로 생성 가능한 개체이다. Hou의 알고리즘에서는 초기세대를 생성할 때, 높이 정보를 이용하여 노드간의 선행관계를 유지하는 개체를 임의로 생성하게 된다. 높이가 1인 n_1을 시작으로 높이가 2에 해당하는 n_2나 n_3 중에서 임의로 한 노드를 선택하여 집단화한다. 만약 n_2를 선택하면 n_4와 같은 집단으로 묶여 (a)와 같은 스케쥴이 생성되고, n_3를 선택하면 역시 n_4와 같이 집단화되어 (b)와 같은 스케쥴이 생성된다.

초기세대에서 생성된 개체에 대하여 교배연산을 수행하는 경우, Hou의 알고리즘에서는 변이가 발생할 노드와 동일한 높이 정보를 갖는 노드를 서로 교환함으로써 새로운 개체를 생성하게 된다. 그림에서 살펴보면 새로운 개체 생성에 영향을 미치는 노드는 n_2와 n_3이다. 만약 그림 (a)에서 n_2가 변이할 노드라고 하면, n_2의 높이 정보가 2이고 동일 높이 정보를 갖는 노드는 n_3이므로, n_2와 n_3를 교환하여 새로운 개체를 생성하게 되나, 그 결과는 새로운 개체가 아닌 초기세대부터 존재하는 그림 (b)에 해당하는 개체가 된다. 또한 n_3가 변이 노드라고 할 경우에도 높이 정보가 2가 되는 n_2와 교환하게 됨으로, 역시 그림 (a)에 해당하는 개체를 다시 생성하게 된다. 따라서 그림 11의 DAG에서 Hou의 알고리즘은 변이연산을 통하여 새로운 개체를 생성할 수 없다.

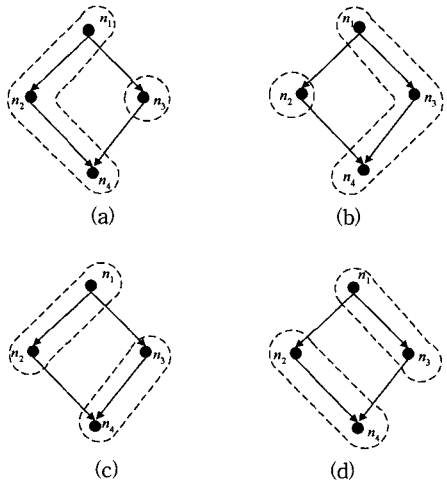


그림 11 다이아몬드 형태를 갖는 DAG의 스케쥴링

제안된 알고리즘에서는 초기세대에서 n_1을 시작으로 직접 후임노드 중에서 임의로 노드를 선택하면서 개체를 생성하게 된다. n_1의 직접 후임노드 중에서 n_2를 선택하게 되면 n_1, n_2, n_4가 동일 프로세서에 할당되는 그림 (a)와 같은 스케쥴이 생성된다. n_1의 직접 후임노드

중에서 n_3를 선택하게 되면 n_1, n_3, n_4가 동일 프로세서에 할당되어 그림 (b)와 같은 스케줄링이 생성되어 Hou의 알고리즘과 방법은 다르나 결과적으로 동일한 초기 세대를 구성하게 된다. 그러나 제안된 알고리즘에서는 변이연산을 통해 그림 (c)와 (d) 형태의 스케줄을 생성할 수 있다. 변이 작용은 Hou의 알고리즘과 달리 4개의 노드 어디에서나 일어날 수 있고, 변이가 일어나는 노드에서는 해당 노드의 직접 전임노드나 직접 후임노드의 집단화를 변화시킴으로써 새로운 개체를 생성하게 된다. 그림 (a)의 경우 n_1에서 변이가 일어난다고 가정하면 n_2와의 결합대신 또다른 직접 후임노드인 n_3와 동일한 집단을 구성하게 되므로 그림 (d)와 같은 형태의 새로운 개체를 생성할 수 있게 된다. 뿐만 아니라 n_4에서 변이가 발생한다면 n_2대신 또다른 직접 전임노드인 n_3와 동일 집단으로 묶여 그림 (c)와 같은 형태의 개체가 생성된다. 그림 (b)의 경우에도 그림 (c)와 (d)의 개체를 변이연산을 통하여 얻을 수 있으므로, 본 논문에서 제안한 유전 알고리즘은 가능한 형태의 모든 스케줄을 고르게 생성할 수 있다는 장점을 가지고 있다.

4. 시뮬레이션

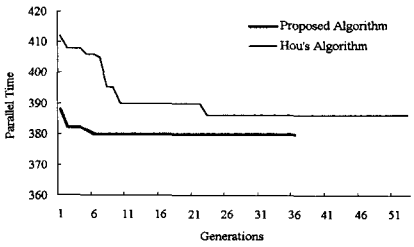
본 논문에서는 유전 알고리즘을 이용한 선형 스케줄링 방법의 성능을 분석하고 기존의 알고리즘과 비교하기 위하여 시뮬레이션을 수행하였다. 유전 알고리즘을 이용한 대표적인 스케줄링 방법인 Hou의 알고리즘을 비교대상으로 선택하였고, 시뮬레이션은 ANSI C로 작성하였으며 Visual C++ 4.2에서 컴파일하여, Pentium 200MHz PC에서 실행하였다.

시뮬레이션에 사용되는 파라메터로는 작업노드의 실행비용과 링크간의 통신비용으로 하였으며, DAG의 작업노드의 수는 25, 50, 100, 200개로 변화시켜가며 스케줄링을 수행하였다. 그래프의 노드당 최대 링크 수는 3에서 10사이로 하였고, 각 노드가 가지는 최대 작업 실행비용은 100으로 하였다. 각 작업 그래프는 난수 발생기를 사용하여 노드의 위치와 링크의 연결도, 링크에 따른 통신비용을 정하였다. 시뮬레이션에서는 각 세대의 개체수를 20, 최대 세대수를 500으로 하였으며, 동일한 엘리트 해(elite solution)가 30세대 이상 계속되는 경우 최종해로 수렴한 것으로 가정하였다. 각 세대의 개체수를 작게하는 경우 시뮬레이션 시간은 줄어드나 다양한 개체가 생성되지 않아 최종해로 수렴하는데 필요한 세대수가 증가하며, DAG를 구성하는 노드의 개수가 증가할수록 그 세대수는 급격히 늘어난다.

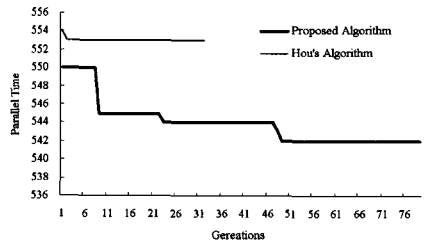
교배나 변이 발생확률을 일정한 값 이상 설정하면 알고리즘의 수행시간과 시뮬레이션 시간은 길어지나 그 결과는 더 좋아지지 않으며, 또한 너무 낮게 설정하면 다양한 스케줄 생성이 어려워진다. 본 논문에서는 적절한 교배 발생확률과 변이 발생확률을 시뮬레이션을 통하여 얻었으며, 각각 0.6, 0.3으로 설정하였다. 변이 발생확률은 다른 경우와 달리 치사 유전자를 만들 가능성이 전혀 없고 다양한 스케줄을 생성하는데 기여하는 바가 크므로 다른 논문에 비하여 확률을 상대적으로 높게 정하였다. 또한, 일부 작업노드의 수행시간의 변화에 따라 발생될 수 있는 파인 그레인 DAG에 대하여 유전 알고리즘을 사용한 선형 스케줄링 방법이 효과적인지 분석하기 위해 각각 10%, 20%, 30%에 해당하는 노드의 실행비용을 임의로 변경한 후 이전의 스케줄링 결과를 이용해 재스케줄링(rescheduling)하였다.

우선 노드의 개수가 25, 50, 100, 200개인 임의의 코오스 그레인 DAG를 발생시킨 후 제안된 알고리즘과 Hou의 알고리즘을 적용하여 작업 완료시간을 비교하였다. 동일 DAG에 대하여 5회 반복 실험한 후 수렴한 세대수와 작업 완료 시간을 구하였다. 그림 12(a)는 작업노드가 25개로 구성된 DAG에 대한 시뮬레이션 결과를 그래프로 나타낸 것이다. 제안된 알고리즘은 10세대 미만에서 작업 완료 시간의 감소가 일어난 후 36세대에서 수렴하였으나, Hou의 알고리즘에서는 20세대까지 작업 완료 시간 감소를 보인 후 50세대가 지나서 수렴하였다. 그림 12(b)은 50개의 노드로 이루어진 DAG에 대해 결과로, 제안된 알고리즘은 10, 20, 50세대 근처에서 작업 완료 시간의 현저한 감소가 발생한 후 최종해로 수렴하였다. Hou의 알고리즘의 경우 2세대에서 작은 변화가 생긴 후 감소없이 수렴하였다. 그림 12(c)는 100개의 노드로 구성된 DAG로부터 얻은 결과로, 제안된 알고리즘은 15세대에서 급격히 감소 후 45세대에서 수렴하였으며, 그림 12(d)의 경우 제안된 알고리즘은 200개의 노드에 대해 10세대와 35세대 근처에서 현저히 감소한 후 60세대에서 수렴하였다. 제안된 알고리즘은 대부분 50세대 미만의 빠른 수렴 속도를 보이고 있으나, Hou의 알고리즘을 적용하는 경우에는 수렴속도가 현저히 느려지거나 작업 완료시간이 길어지며, 100개와 200개의 노드로 구성된 DAG에서는 거의 100 세대 근처에서 수렴하는 경우도 나타났다. 각 세대에서 가장 우수한 개체를 선택하는 엘리트 보존 정책을 사용하였기 때문에 두 방법의 작업 완료 시간간의 차이는 크지 않으나, 제안된 알고리즘에서는 다양한 개체를 발생시키므로 평균적인 개체의 질은 훨씬 더 우수하다.

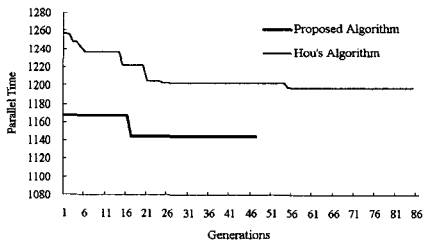
그림 13부터 그림 14는 임의로 생성한 DAG에서 각



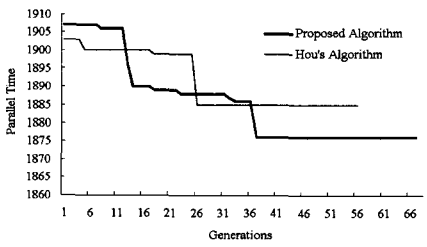
(a) 작업노드가 25개인 경우



(b) 작업노드가 50개인 경우



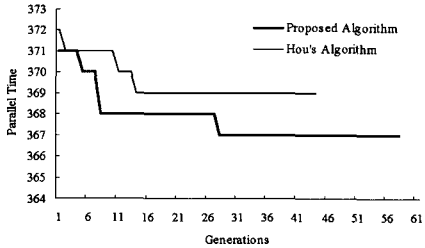
(c) 작업노드가 100개인 경우



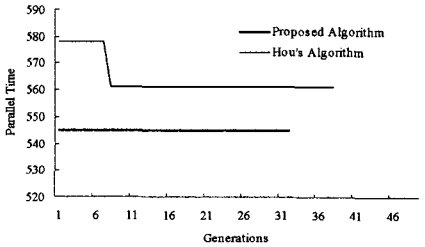
(d) 작업노드가 200개인 경우

그림 12 각 세대별 작업 완료 시간의 비교

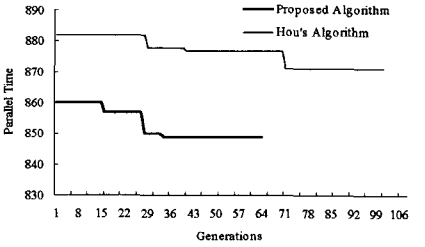
각 전체 노드의 10%, 20%, 30%에 해당하는 노드에 대하여 작업비용을 변화시킨 후 이전의 스케쥴링 정보를 이용하여 재스케쥴링한 결과를 그래프로 표현한 것이다. 그림 13은 노드의 개수가 25, 50, 100, 200개인 DAG에



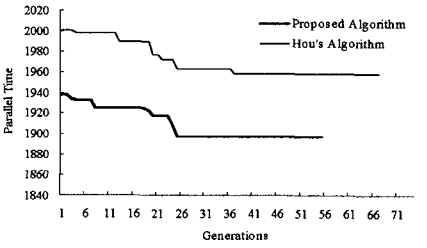
(a) 작업노드가 25개인 경우



(b) 작업노드가 50개인 경우



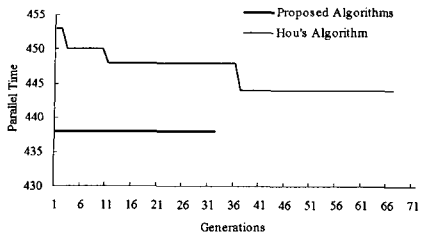
(c) 작업노드가 100개인 경우



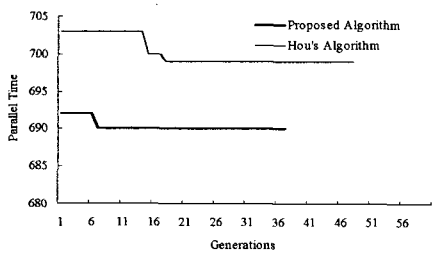
(d) 작업노드가 200개인 경우

그림 13 DAG에서 10%에 해당하는 노드의 작업 실행 비용을 변화시킨 경우 각 세대별 작업 완료 시간의 비교

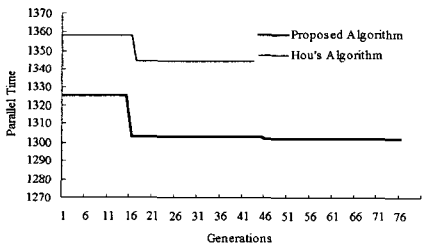
서 10%에 해당하는 노드의 작업 실행비용을 변화시킨 후 다시 스케쥴링하여 얻은 두 알고리즘의 작업 완료 시간을 각 세대별로 비교한 것이다. 그림 (a)는 노드 수가



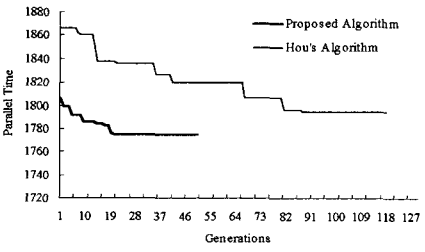
(a) 작업노드가 25개인 경우



(b) 작업노드가 50개인 경우

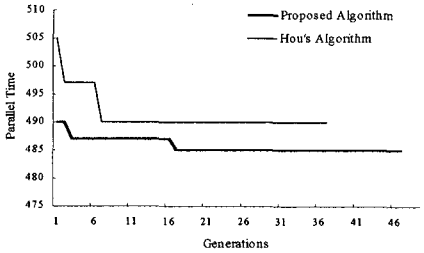


(c) 작업노드가 100개인 경우

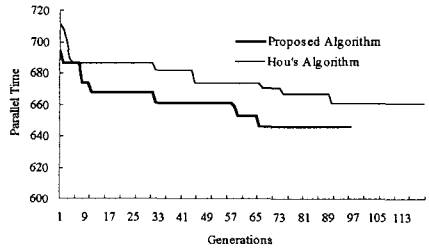


(d) 작업노드가 200개인 경우

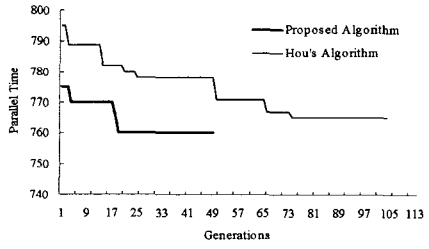
그림 14 DAG에서 20%에 해당하는 노드의 작업 실행 비용을 변화시킨 경우 각 세대별 작업 완료 시간의 비교



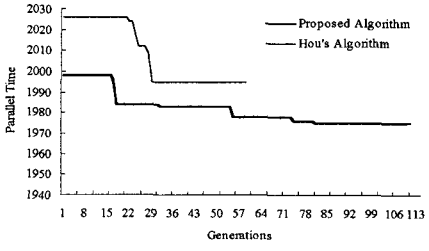
(a) 작업노드가 25개인 경우



(b) 작업노드가 50개인 경우



(c) 작업노드가 100개인 경우



(d) 작업노드가 200개인 경우

그림 15 DAG에서 30%에 해당하는 노드의 작업 실행 비용을 변화시킨 경우 각 세대별 작업 완료 시간의 비교

25개인 DAG에 대해 시뮬레이션한 것으로 제안된 알고리즘은 10세대 미만과 25세대 근처에서 작업 완료 시간이 급격히 감소한 후 40세대에서 수렴하였으나, Hou의 알고리즘은 10세대 부근에서 감소한 후 45세대에서 수렴하였다. 그림 (b)는 50개의 노드로 구성된 DAG에 대한 결과로 제안된 알고리즘에 의해 초기세대에서 생성된 엘리트

해가 30세대까지 변화없이 나타났으며, 이것은 초기세대에 가장 우수한 개체가 발생되었기 때문이다. 100개의 노드에 해당하는 그림 (c)에서는 제안된 알고리즘에 의하여 생성된 초기세대의 해가 15 - 35 세대에서 감소한 후 50세대에서 수렴하였다. 그림 (d)에서 제안된 알고리즘은 25세대까지 여러 번의 현저한 감소를 보인 후 56세대 근처에서 수렴하였다. 역시 제안된 알고리즘은 대부분의 경우 50세대 이전에 최종해로 수렴하였으며 작업 완료 시간에서도 Hou의 알고리즘에 비해 우수함을 보여주고 있다.

그림 14와 그림 15는 노드의 개수가 25, 50, 100, 200개인 DAG에서 20%와 30%에 해당하는 노드의 작업 실행 비용을 임의로 변화시킨 후 다시 스케쥴링한 결과를 비교한 것이다. 주목할만한 사항으로 그림 14(d)에서 제안된 알고리즘은 20세대까지 작업 완료 시간이 점차 감소한 후 45세대에서 수렴하였으나, Hou의 알고리즘은 90세대까지 감소한 후 120세대에서 수렴하였다. 그림 15(c)의 경우 제안된 알고리즘은 3세대와 15세대에서 작업 완료 시간이 감소한 후 45세대에서 수렴하였으나, Hou의 알고리즘은 10세대, 50세대, 60세대 부근에서 감소한 후 110세대에서 수렴하였다.

5. 결 론

다중프로세서 시스템은 일반적으로 하나의 프로세서에서 수행되는 프로그램을 작은 모듈로 분할하여 여러 개의 프로세서에 할당하여 수행함으로써 매우 높은 성능 향상을 얻는 방법이다. 그러나 이런 다중프로세서 시스템에서는 분할된 모듈이 효율적으로 실행되도록 스케쥴링 하여야 하며 각 프로세서에 가급적 균일한 부하가 주어지도록 해야한다. 많은 프로세서를 사용한다고 항상 성능이 향상되는 것은 아니며, 스케쥴링이 적절히 이루어지지 않으면 단일프로세서를 사용하여 프로그램을 수행하는 것 보다 성능이 떨어지는 경우도 생긴다. 본 논문에서는 다중프로세서 시스템에서 유전 알고리즘과 DAG의 특성이 고려된 선형 집단화를 이용하여 스케쥴링하는 방법을 제시하였고, 시뮬레이션에서 제안된 알고리즘을 무작위로 생성한 DAG에 적용해 사용 가능성을 검증하였다.

코오스 그레인 DAG에서는 선형 집단화가 비선형 집단화보다 항상 우수한 스케쥴링 방법이 될 수 있음을 이용해, 본 논문에서는 선형 집단화를 보장하는 유전 알고리즘을 제한하였다. 현재 노드를 기준으로 직접 후임 노드만을 선택하는 집단화 방법으로 초기세대를 생성하며, 유전연산에서도 직접 전임노드나 직접 후임노드만을 선택하여 선형성을 유지하였다. 시뮬레시션 결과 대부분의 DAG에서 50세대 내외의 빠른 수렴을 보였으며, 최종 해에 대해서도 Hou의 알고리즘을 통한 결과보다 작업 완료 시간이 감소하였다. 물론 이러한 선형 집단화는 모든 DAG에 대하여 항상 최적의 스케쥴링을 제공하지는 않지만, 프로그램의 병렬성을 최대한 활용할 수 있을 뿐 아니라, 각 노드의 실행비용이나 통신비용의 변화에 비교적 민감하지 않은 장점을 가지고 있다.

추후 연구과제로 제안된 유전 알고리즘을 실제 다중프로세서 시스템에 적용하여 실시간 사용 가능성을 검증할 예정이며, 파인 그레인을 나타내는 DAG에 대하여 빠르게 수렴할 수 있는 비선형 집단화가 가능한 유전자 표현방법과 유전 연산자를 고안하여 선형 집단화를 이용한 유전 알고리즘과 비교할 계획이다.

참 고 문 헌

[1] K. Hwang, Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

[2] H. El-Rewini, H. H. Ali, and T. Lewis, "Task scheduling in multiprocessing systems," *IEEE Computer*, pp. 27-37, Dec. 1995.

[3] A. Gerasoulis and T. Yang, "On the granularity and clustering of directed acyclic task graphs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 6, pp. 686-701, June 1993.

[4] H. Stone, High-Performance Computer Architectures, Addison-Wesley Company, 1987.

[5] H. Kasahara and S. Narita, "Practical multiprocessing scheduling algorithm for efficient parallel processing," *IEEE Trans. Comput.*, vol. C-33, no. 11, pp. 1023- 1029, Nov. 1984, .

[6] H. Kasahara and S. Narita, "Parallel processing of robot-arm control computation on a multimicroprocessor system," *IEEE J. Robotics Automation*, vol. RA-1, no. 2, pp. 104-113, June 1985.

[7] C. L. Chen, C. S. G. Lee, and E. S. H. Hou, "Efficient scheduling algorithm for robot inverse dynamics computation on a multiprocessor system," *IEEE Trans. Syst., Man, Cybernetics*, vol. 18, pp 729-743, Dec. 1988.

[8] B. Hellstrom and L. Kanal, "Asymmetric mean-field neural networks for multiprocessor scheduling," *Neural Networks*, vol. 5. pp. 671-686, 1992.

[9] N. Ono and T. Tsugawa, "A genetic algorithm for the genetic crew scheduling problem," *International Conference on Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag Wien New York, pp. 270-274, 1998.

[10] J. Jozefowska, R. Rozycki, and J. Weglarz, "A

genetic algorithm for some discrete-continuous scheduling problem," *International Conference on Artificial Neural Nets and Genetic Algorithms*, Springer-Verlag Wien New York, pp. 273-276, 1995.
[11] E. Ramat, G. Venturini, C. Lente, and M. Slimane, "Solving the multiple resource constrained project scheduling," *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 489-496, Jul. 1997.
[12] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113-120, Feb. 1994.
[13] B. Soucek and The IRIS Group, Dynamic, Genetic, and Chaotic Programming: The Six Generation, John Wiley & Sons, Inc., pp. 339-352, 1992.
[14] P. -C. Wang and W. Korfhage, "Process scheduling using genetic algorithms," *Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing*, pp. 638-641, Oct. 1995.
[15] R. C. Correa, A. Ferreira, and P. Rebreyend, "Integrating list heuristics into genetic algorithms for multiprocessor scheduling," *Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing*, pp. 462-469, Oct. 1996.
[16] M. Schwehm and T. Walter, "Mapping and scheduling by genetic algorithms," *Parallel Processing: CONPAR 94 - VAPP VI*, Springer-Verlag, pp. 832-841, Sep. 1994.
[17] M. R. Garey and D. S. Johnson, Computers And Intractability: A Guide to the Theory of NP-Completeness, W. H. FREEMAN, 1979.
[18] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing Company, 1989.
[19] L. Davis, Handbook Of Genetic Algorithms, Van Nostrand Reinhold, 1991.

배 성 환

1996년 인하대학교 전자공학과 (학사). 1998년 인하대학교 전자공학과 (석사). 관심분야는 유전 알고리즘, 다중 프로세서 시스템, 스케줄링

최 상 방

1981년 한양대학교 전자공학과 졸업. 1988년 University of Washington 석사. 1990년 University of Washington 박사. 1981년 ~ 1986년 LG 정보통신(주) 근무. 1991년 ~ 현재 인하대학교 전자공학과 부교수. 관심분야는 컴퓨터구조, 병렬 및 분산처리 시스템, Fault-tolerant computing, 컴퓨터 네트워크