효율적인 멀티프로세서 스케줄링을 위한 유전자 알고리즘 설계

박월선*, 박상일*, 남은미**, 윤성대*
*부경대학교 전자계산학과
**부경대학교 전산교육학과
e-mail:wspark@unicorn.pknu.ac.kr

Genetic Algorithms for Efficient Multiprocessor Scheduling

Weol-Seon Park*, Sang-Il Park*, Eun-mi Nam**, Sung-Dae Youn*
*Dept. of Computer Science, Pukyung National University
**Dept. of Computer Science Education, Pukyung National University

요 약

본 논문은 NP-complete문제중의 하나인 순서제약이 있는 병렬프로그램을 멀티프로세서 시스템 상에서 효율적으로 분배하기 위한 유전자 알고리즘 설계 방법을 제안한다. 순서제약 조건을 만족하게 하는 새로운 염색체 코딩방법 및 휴리스틱한 스케줄링 알고리즘으로 정법한 해를 생성하고 프로세서 효율성을 고려한 평가 함수(evaluation function)와 우수한 유전인자를 이용하여 교배하는 교배연산자 등을 제안하였다. 그리고 제안한 알고리즘을 실험한 결과, 순서제약이 있는 다양한 형태(topology)의 병렬프로그램 스케줄링 문제에 대해서 제안한 유전자 알고리즘의 타당성을 확인하였다.

1. 서론

오늘날 컴퓨터 성능이 기가플롭스(gigaflops)단위로 1초당 1조 번의 부동점 연산을 수행하는 슈퍼컴퓨터의 개발로 시간이 많이 걸리는 NP-complete문제(Non-deterministic Polynomial-complete)에 대해 동일한 성능을 가지는 멀티프로세서의 병렬처리로 시간을 줄일 수 있게 되었다. 그러나 이러한 멀티프로세서의 병렬계산에 대한 요구가 점점 가중됨에 따라 스케줄링 문제가 중요한 논점이 되고 있다[1][2].

병렬처리의 스케줄링 문제는 작업(task)사이에서의 선행 제약 조건을 나타내는 작업그래프(task graph)형태, 다중 프로세서 시스템(multi-processor system)형태, 태스크 처리시간 등을 고려해야 하므로 NP-complete 문제지만 제출된 작업의 응답기간을 줄이고 사용된 프로세서의 수를 줄임으로써 경제적으로 비용을 감소시키고 프로세서를 효율적으로 이용할 수 있는 스케줄링 문제에 관한 알고리즘이 많이 연구되고 있다[1]-[4]. 이러한 기존의 스케줄링 문제의 해결 방안은 대체적으로 list heuristic한 방법으로 준최적 해(suboptimal solution)를 구한다. 그러나 이러한 방법은 다소 문제가 특정적이며 제한적인 가정(즉, 가상적 지식이나 유도되는 지식)을 필요로 한다[2]. 따라서 스케줄링에 관한 다른 대안은 다양한 탐색과 조합 최적합 타입 문제(Combinatorial

-optimization type)에 부각되고 있는 meta-heuristic으로 알려진 유전자알고리즘(Genetic Algorithm : GA)에 대해 많은 연구를 하고있다[5]-[8].

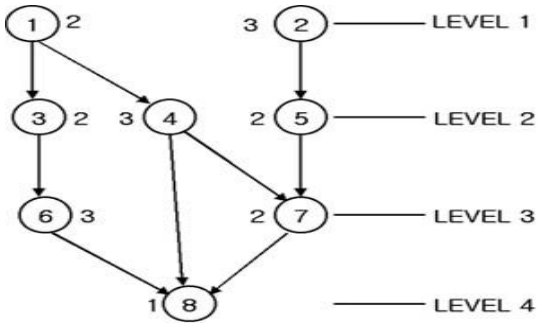
이 유전자 알고리즘은 적자생존 또는 자연선택에 의해 자연계가 진화해왔다는 가설에 기반한 문제해결 기법으로써 랜덤하게 생성된 초기 해(염색체)들로부터 교배(crossover)와 돌연변이(mutation)라는 간단한 유전자 연산자를 통해 계속 진화하다가 해를 요구하는 시점에서 그때까지 진화된 가장 우수한 해를 산출하는 방법이다.

본 논문에서는 pure GA의 변형인 '유전자알고리즘이 기반이 된 선행제약이 있는 병렬 프로그램을 멀티프로세서로 스케줄링 하는 기법'을 제안한다. 이 기법은 초기해 구성(initialization)시 정법한 해를 생성하는 새로운 염색체 코딩방법 및 스케줄링 방법, 세대를 재구성하기 위하여 우수한 해를 판단 및 선택하는 방법, 해의 유전인자 중에서 우수한 유전인자는 그대로 종속시키면서 교배한 후 새로 생성된 교배해가 기존 해보다 더 적합할 가능성이 있을 때만 다음 세대로 유전시키는 교배연산자 등에서 기존 방법과는 다르다.

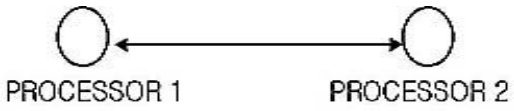
본 논문의 구성은 2장에서 스케줄링 문제를 분석하고 3장에서는 스케줄링 문제에 대해서 제안한 유전자 알고리즘을 문제에 적용하는 방법을 기술하고 4장에서는 제안된 유전자의 유용성 및 실용성을 보이기 위해 선행제약(순서제약)이 있는 병렬 프로그램의 다양한 형태(topology)에서 실험한 결과를 보이고 마지막으로 5장에서 결론을 맺는다.

2. 병렬프로그램 스케줄링 문제

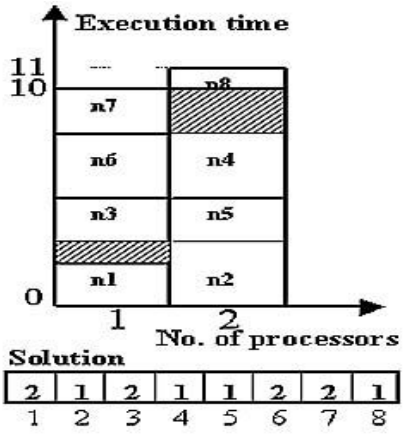
그래프와 방향 그래프는 전산과학, 전기 공학, 경제학, 수학, 물리학, 화학, 통신, 게임 이론, 그리고 많은 다른 분야에서의 수많은 문제와 구조에 대한 유용한 추상적 개념이라 할 수 있다.



(그림1) 병렬프로그램 그래프



(그림2) 멀티프로세서 상호관계



(그림 3) 염색체에 따른 스케줄링

본 논문에서는 스케줄링 해야 할 문제를 방향성과 가중치가 있는 그래프(DAG)로 나타냈으며 하나의 노드에 그 자신을 연결하는 간선이 존재하지 않는다는 것과 그래프에서는 한 쌍의 노드들 사이에 같은 방향을 갖는 두 개의 간선이 존재할 수 없다는 것과 간선에 의해 연결된 노드들이 사이클(cycle)을 이루지 않는다는 것을 함축하고 있다.

그림1은 계산될 병렬 프로그램의 한 부분을 일반화하여 나타낸 그래프(DAG)이며, 기호로 표현하면 G = (V, E, W)가 되며, 여기서 V는 처리될 태스크(task)의 집합으로써 원형의 노드로 표시하고, E는 처리될 태스크사이의 선행 제약 조건들의 집합으로써 방향을 나타내는 간선으로 표시하고 W는 태스크의 처리시간(cost)을 범위 1에서 4까지 랜덤하게 발생시켜 노드 옆에 수치로 나타냈으며 각 태스크의 번호는 그래프의 단계(level)에 따라 왼쪽에서 오른

쪽 순서대로 붙여진다.

따라서 그림1의 의미를 예를 들어 설명하면 다음과 같다. 만약 한 프로세서가 태스크 1과 3을 처리한다면, 노드1과 3은 단계가 다르고 노드 1과 3 사이에 간선이 존재하므로 노드3은 노드1이 먼저 수행된 다음 실행할 수 있고, 처리시간은 노드1이 2이고 노드3도 2이므로 노드 1에서 노드 3까지 수행하는데 걸리는 시간은 4가 된다. 또한, 같은 단계에 있는 태스크들은 우선 순위가 동등하므로 같은 프로세서상에서 처리 할 때 각각의 노드에 연결된 선조 노드들이 먼저 실행되었다면 노드의 순서에 상관없이 처리하며 또한 그래프상의 모든 노드들은 한번씩 중복없이 수행되어야 하며 프로세서들에 의해 병렬로 처리 될 때 전체 걸린 수행시간은 적을수록 좋은 결과이다. 그림 2는 멀티프로세서 시스템 상에서 프로세서간의 상호 연결 망 형태를 보이며 그림 3은 각 프로세서에 스케줄링된 태스크들을 처리하는데 걸리는 시간이 적어도 11이 필요함을 나타낸다.

3. 유전자 알고리즘 설계

선행 제약조건이 있는 다양한 형태(topology)의 병렬 프로그램을 멀티프로세서 시스템상의 병렬 프로세서들로 스케줄링 제약조건에 위배되지 않게 효율적으로 분배해서 각 프로세서들의 응답시간을 감소시키고 시스템의 처리능력을 향상시키는 스케줄링 결과 값(suboptimal solution or the best solution)을 유전자 알고리즘을 이용하여 검색하는 방법을 설명한다. 먼저 유전자 알고리즘을 어떤 문제 해결에 사용하기 위해서는 해를 염색체 형태로 코드화 하는 방법(chromosome representation), 새로운 염색체를 생성시키는 유전 연산자인 교배(crossover)와 돌연변이의 방법(mutation), 염색체에 의해 표현되는 해의 적합도를 측정하는 평가함수(evaluation) 그리고 모집단(population)을 구성하는 방법(reproduction)이 고안되어야 한다. 위 요소들이 어떻게 구성되어지는가에 따라 유전자 알고리즘의 특성과 성능이 결정된다.

제안한 유전자 알고리즘은 다음과 같다.

<Procedure advanced genetic algorithm>

Step 1 :

Read the DAG and build a database which includes the adjacency list(node n × n), the cost list, the rate of crossover(Pc), the rate of mutation(Pm), the number of processors (Pro _num), the size of populations(Pop_size), the number of generations(T) and so on

Step 2 :

Randomize the chromosome based on the sort of processors to generate an initial population of size(Pop_size).

Step 3 :

For i=2 to T

Apply the scheduling function and compute fitness value with evaluation function.

Select chromosomes based on the modified roulette wheel.

Execute GA operators, Crossover and mutation, for each ratio.

end for

Step 4 :

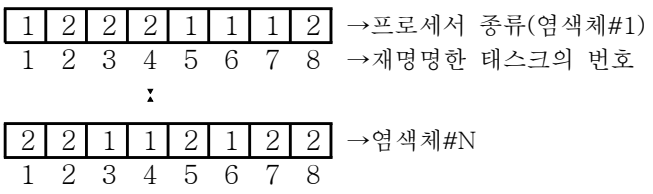
Select the best solution at the current generation.

단계1에서는 컴퓨터에서 처리할 수 있도록 문제를 읽어서 데이터베이스형태로 만든다. 단계2에서는 첫 번째 세대를 구성하기 위하여 초기 해(initialize solution)를 Pop_size 만큼 랜덤하게 발생한다. 단계3은 실질적인 처리를 하는 단계로써, 종료세대가 될 때까지 스케줄링 함수에 의해 태스크를 프로세서에 분배한 후, 각 해가 문제의 목적에 어느 정도 부합되는가를 계산하고 변형된 룰렛 휠(roulette wheel) 방법으로 Pop_size 만큼 해를 선택해서 다음세대를 구성한 후, 새로 구성된 세대의 각 염색체에 대해서 유전자 연산자인 교배와 돌연변이를 각각의 비율만큼 수행한다. 단계4는 종료세대인 현재세대에서 가장 좋은 해를 산출한다.

3.1 초기 모집단 구성 (Initialization)

기존의 많은 연구에서는 스케줄링제약 조건에 위배되지 않게 해를 구성하기 위해서 선조의 수(predecessor)나 계승자의 수(successor)를 계산한 정보를 기반으로 하여 해를 구성하지만[5]-[8], 본 논문에서 제안된 염색체 코딩방법은 쉽고 간단하게

그림1과 같이 병렬 프로그램을 나타내는 그래프의 level에 따라 태스크의 번호를 재명명(rename)한 후, 태스크 속성에는 태스크를 처리할 프로세서의 종류로 표시하는 방법을 도입함으로써 같은 프로세서 내에서 선조를 처리하기 전에 자신을 처리하는 불법의 해(illegal solution) 또는 비가능 해(infeasible)를 생성하지 않는다. 따라서 초기 인구를 구성하기 위하여 모집단의 크기만큼 사용될 프로세서의 종류를 그림4와 같이 랜덤하게 생성한다.



(그림 4) 초기 모집단 구성

3.2 평가 (Evaluation)

각 염색체의 적합도를 평가하기 이전에 다른 프로세서 상에서의 위법한 해 또는 불가능한 해를 생성하지 않도록 스케줄링 한다. 즉, 그림 4의 염색체#1에서 2번 프로세서가 8번 태스크를 수행하려면 8번 태스크의 선조 태스크들인 4, 6, 7번이 이미 수행되었는지를 체크한 후 선조 태스크 중 한 개의 태스크라도 수행되지 않았다면 선조 태스크 모두가 수행될 때까지 기다린 후 8번 태스크를 그림 3과 같이 스케줄링 한다. 그 다음 각 프로세서의 실행시간(makespan)에 대해 다음과 같이 제안된 평가 함수에 의해 fitness value를 구한다.

프로세서의 효율성을 고려하기 위하여 염색체를 스케줄링 했을 때 사용된 프로세서 중 제일 긴 실행시간을 가진 프로세서(max(pms_i))에 대해서 제일 짧은 실행시간을 가진 프로세서(min(pms_i))의 시간비율이 0.5보다 작으면 파라메터 α값을 적절하게 변화시켜 다음 세대로 유전될 확률을 감소시키며, 모집단에서 제일 긴 실행시간을 가진 염색체라도 여러 번의 세대교체가 일어난 후에는 좋은 해가 될 가능성이 있음을 고려해서 적합도 함수는 식(1)과 같이 구할 수 있다.

$$Fitness = \alpha \times (\beta - \max(pms_i)) \qquad (1)$$

$$Where, ① \left(\begin{array}{ll} \text{if } (\frac{\min(pms_i)}{\max(pms_i)} < \frac{1}{2}), & \alpha = 100 \\ \text{otherwise}, & \alpha = 10 \end{array} \right)$$

$$② \quad pmax = \sum_{i=1}^{n} \max(\max(pms_i))$$

$$\beta = pmax \times 0.001 + pmax$$

여기에서 구한 Fitness value에 따라 현재의 해가 스케줄링의 목적에 어느 정도 적합한가를 판단하는 기준이 되며 fitness value가 클수록 좋은 해다.

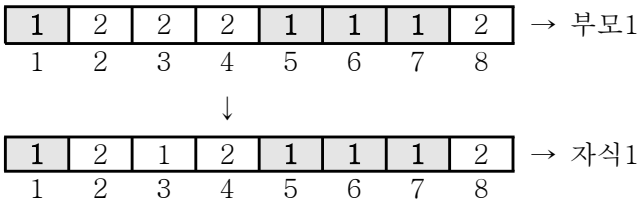
3.3 선택 (Selection)

적자생존 및 자연선택의 방법에 따라 자연환경에 잘 맞는 생명체만 계속 다음 세대로 유전되어 진화하듯이 우수한 해를 선택하여 다음 세대로 이전시키기 위해서 제안한 선택기법은 Fitness value를 내림차순으로 정렬한 다음 제일 좋은 해를 일부 선택하고 나머지는 룰렛 휠(roulette wheel)방법[6]으로 새로운 모집단을 구성한다.

3.4 유전자 연산자

3.4.1 교배 (Crossover)

제안된 교배방법은 새로운 염색체를 생성하기 위해서 교배비율만큼 랜덤하게 부모1, 2를 선택한 후 부모1에서 실행시간이 가장 적은 프로세서의 유전자의 위치(우성 유전인자)를 자식1에게 유전시킨 후 사용된 프로세서의 종류를 랜덤하게 발생하여 나머지 유전인자를 결정한다. 자식2도 부모2에서 위와 같은 과정으로 생성한다. 즉, 그림 4의 염색체 #1을 부모1로 가정하면 그림 3과 같이 사용된 프로세서 실행시간을 알 수 있으므로 자식1이 그림5와 같이 생성된다.



(그림 5) 자손염색체 생성과정

교배를 통해 새로 생성된 염색체는 무조건 다음세대로 전이되는 것이 아니라 빠른 수렴을 위해서 우수한해가 될 수 있는 가능성이 있을 때만 유전된다.

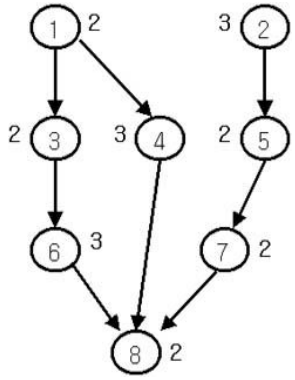
3.4.2 돌연변이 (Mutation)

돌연변이방법은 더 넓은 해 공간을 가지고 최적해를 탐색할 수 있도록 모든 유전인자에 대하여 랜

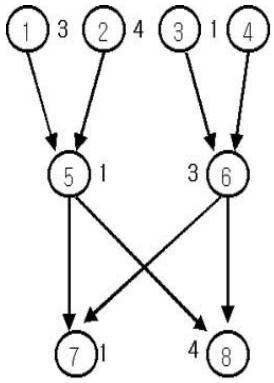
덤하게 임의의 수를 발생했을 때 그 수가 돌연변이 비율보다 작은 값이면 그 위치의 유전인자에 프로세서의 종류를 랜덤하게 발생하여 대체한다. 돌연변이 된 유전인자를 가진 새 염색체도 빠른 수렴을 위해서 우수한 해가 될 수 있는 가능성이 있을 때만 유전되게 한다.

4. 모의 실험(Simulation)

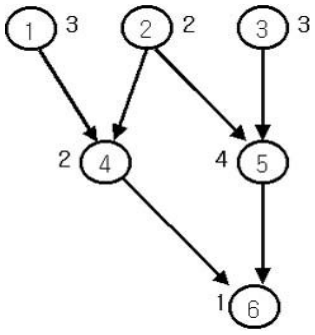
제안된 알고리즘의 유용성과 실용성을 보이기 위해 그림 6, 7, 8과 같은 선행 제약이 있는 병렬 프로그램의 다양한 형태(topology)에 대해서 실험한다.



(그림 6) 유형 1



(그림 7) 유형 2



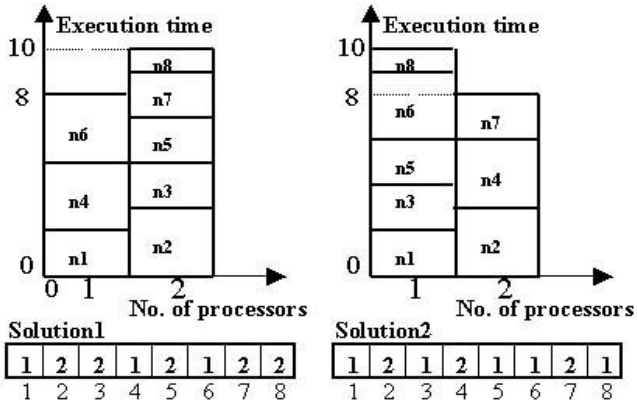
(그림 8) 유형 3

그림 6, 7, 8에서 보인 각 유형의 태스크 그래프를 제안한 유전자 알고리즘으로 효율적인 스케줄을 탐색하기 위해 사용된 GA 파라메터는 표1과 같다.

<표 1> GA 파라메터

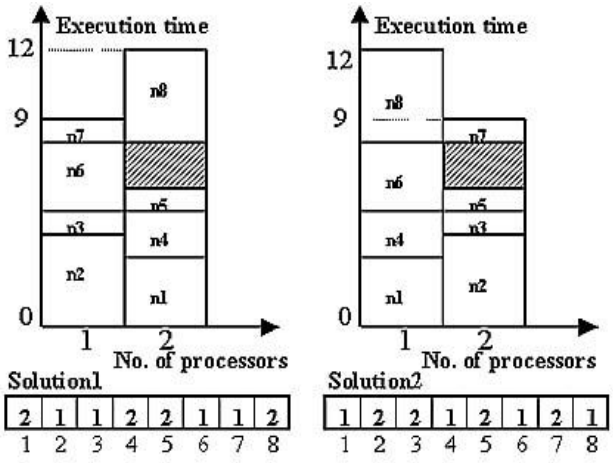
파라메터 명	파라메터 값	설명
N-num	6 ~ 8	사용된 태스크의 수
Pop_size	3	모집단의 크기
Pro_num	2	사용된 프로세서 수
Pc	0.4	교배 비율
Pm	0.2	돌연변이 비율

알고리즘을 6회 수행한 결과, 그림9와 그림11은 유형1, 3의 해로 실행시간(makespan)이 10인 두 개의 다른 해를 보이며, 그림10은 유형 2의 해로 실행시간이 12인 두 개의 다른 해를 보인다. 그림10, 11에서의 사선부분은 프로세서의 유휴시간(idle)을 나타낸다. 구한 해는 매우 유효한 해로써 초기 세대에서 수렴하였다.



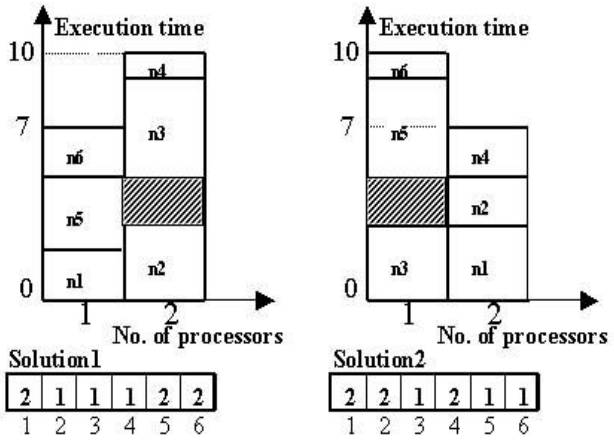
(a) (b)

(그림 9) 유형 1의 스케줄링 결과



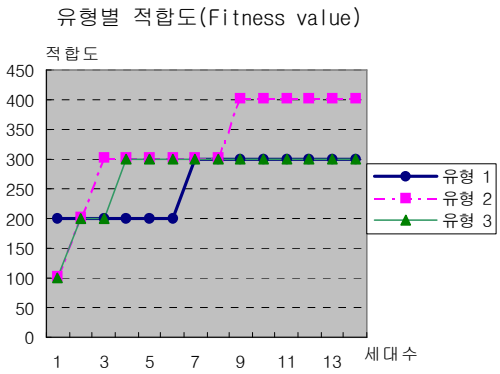
(a) (b)

(그림 10) 유형 2의 스케줄링 결과



(a) (b)

(그림 11) 유형 3의 스케줄링 결과



(그림 12) 각 유형별 적합도

그림 12는 각 유형별 태스크 그래프를 6회 실험한 결과 중에서 늦게 수렴한 경우이며, 태스크 수와 간선 수를 증가시켜 실험한 결과도 세대수가 증가할수록 적합치(Fitness value)가 커지고 실행시간은 반대로 작아지는 이상적인 유전자 알고리즘 해 탐색 형태를 나타내었다.

5. 결론

본 논문에서는 순서제약과 각 태스크를 처리하는데 걸리는 시간인 가중치(weight)가 있는 다양한 형태(topology)의 병렬 프로그램을 멀티프로세서 시스템상의 병렬 프로세서들로 스케줄링 제약조건에 위배되지 않게 효율적으로 분배해서 각 프로세서들의 응답시간을 감소시키고 시스템의 처리능력을 향상시키는 스케줄링 결과 값(suboptimal solution or the best solution)을 검색하기 위해서 유전자 알고리즘을 적용하는 방법을 제안하였다.

제안된 유전자 알고리즘에서는 순서 제약 조건을 위배하지 않기 위해서 기존 연구에서 필요로 하는 휴리스틱한 정보인 각 노드에 연결된 선조 노드들(predecessor)의 수 혹은 계승노드(successor)의 수를 계산하지 않고 새로운 염색체 코딩방법 및 스케줄링 알고리즘으로 정법한 해만을 생성하게 하며, 프로세서의 효율성을 고려한 평가 함수(evaluation function)와 우수한 유전인자를 교배하는 교배연산자(crossover)등을 도입하여 다양한 형태의 병렬 프로그램 일부를 만들어서 파라메터의 값을 바꾸어 가며 실험하였다.

실험 결과 제안된 유전자 알고리즘은 최적해 혹은 준 최적 해를 빠른 시간 내에서 탐색하므로 순서제약이 있는 스케줄링문제에 유용성 및 실용성이 있음을 알 수 있었다.

향후 연구과제로는 일반화된 혹은 임의의 형태의 작업 태스크 그래프가 아닌 특정한 문제 형태 즉, FFT(Fast Fourier Transform)와 GAUSS에서 제안한 유전자 알고리즘을 적용하는 것이다.

참고문헌

[1] H. El-Rewini, T. G. Lewis, and H. H. Ali, Task Scheduling in Parallel and Distributed Systems. 1994

[2] H. El-Rewini, “Partitioning and Scheduling," Parallel and Distributed Computing Handbook, A. Y. Zomaya, ed., pp. 239-273. 1996

[3] H. Kasahara and S. Narita, "Practical Multi-processing Scheduling Algorithms for Efficient Parallel Processing," IEEE Trans. Computers, vol. 33, pp. 1,023-1,029, 1984

[4] B. Bataineh and B. Al-Asir, "Efficient Scheduling Algorithm for Divisible and Indivisible Tasks in Loosely Coupled Multiprocessor Systems," Software Eng. J., pp. 13-18, 1994

[5] Ricardo C. Corrêa,"Scheduling Multiprocessor Tasks with Genetic Algorithms," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 8, Aug. 1999

[6] Albert Y. Zomaya, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues," IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 8, Aug. 1999

[7] E. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling." IEEE Trans. Parallel and Distributed Systems, vol. 5, no. 2, pp. 113-120, Feb. 1994

[8] I. Ahmad and M. K. Dhodhi, "Multiprocessor Scheduling in a Genetic Paradigm", Parallel Computing, vol. 22, pp. 395-406, 1996