분산병렬 시스템에서 유전자 알고리즘을 이용한 스케쥴링 방법

정회원 김 화 성*

Generic Scheduling Method for Distributed Parallel Systems

Hwa-sung Kim* *Regular Member*

요 약

본 논문에서는 고속 네트웍 기반의 분산 병렬 시스템에서 다양한 내재 병렬 형태를 갖는 프로그램의 효과적인 수행을 위한 유전자 알고리즘 기반의 태스크 스케쥴링 방법(Genetic Algorithm based Task Scheduling· GATS)을 제안한다. 분산병렬 시스템은 고속 네트워을 통해 연결되어진 다수의 범용, 병렬, 벡터 컴퓨터들로 구성되어진다. 분산병렬 처리의 목적은 다양한 내재 병렬 형태를 갖는 연산 집약적인 문제들을 다수의 고성능 및 병렬 컴퓨터들의 각기 다른 능력을 최대한 이용하여 해결함에 있다 분산병렬 시스템에서 스케쥴링을 통하여 더 많은 속도향상을 얻기 위해서는 시스템간의 부하 균형보다는 태스크와 병렬 컴퓨터간의 병렬특성의 일치가 주의 깊게 다루어져야 하며 태스크의 이동으로 인한 통신 오버헤드가 최소화되어야 한다 본 논문에서는 유전자 알고리즘의 동작이 병렬 특성을 감안하여 이루어질 수 있도록 초기화 방법과 지식 기반의 mutation 방법을 제안한다

ABSTRACT

This paper presents the Genetic Algorithm based Task Scheduling (GATS) method for the scheduling of programs with diverse embedded parallelism types in Distributed Parallel Systems, which consist of a set of loosely coupled parallel and vector machines connected via high speed networks The distributed parallel processing tries to solve computationally intensive problems that have several types of parallelism, on a suite of high performance and parallel machines in a manner that best utilizes the capabilities of each machine. When scheduling in distributed parallel systems, the matching of the parallelism characteristics between tasks and parallel machines rather than load balancing should be carefully handled with the minimization of communication cost in order to obtain more speedup. This paper proposes the biased initialization methods for an initial population and the knowledge-based mutation methods to accommodate the parallelism type matching in genetic algorithms

I. 서 론

이기종 시스템의 특수한 형태인 분산병렬 시스템은 고속 네트워을 통해 약한 결합 (Loosely Coupled) 형식으로 연결되어진 다수의 범용, 병렬, 벡터 컴퓨터 시스템들로 구성 되어진다[1,2]. 분산병렬 처리의 목적은 다양한 내재 병렬 타입의 연산 집약적인 문제들을 다수의 고성능 및 병렬 컴퓨터들의 각기 다른 능력을 최대한 이용하여 해결함에 있다 많은 종류의 대규모 과학 애플리케이션들은 흔히 하나 이상의 내재 병렬 타입을 갖는 코드 모듈로서 이루어진다. 내재 병렬 타입으로는 SIMD, MIMD 혹은 vector 등이 있다 이와 같은 유형의 애플리케이션들은 단일 병렬 컴퓨터를 통하여서는

* 광운대학교 전자공학부 네트워크 컴퓨팅 연구실(hwkim@daisy kw ac kr),
논문번호 020273-0612 , 접수일자 2002년 6월 12일
※ 본 연구는 광운대학교 2001년도 교내 학술연구비 지원에 의해 수행되었습니다

일반적으로 최적의 속도 향상(Speedup)을 기대하기 어렵다 오히려 서로 다른 병렬 타입의 코드 모듈별로 이들을 가장 잘 처리할 수 있는 서로 다른 병렬 시스템들에 할당하여 수행하는 것이 보다 효과적이다 그러나 코드 모듈들을 여러 시스템으로 할당할 때, 코드 모듈의 이동으로 인해 야기되는 네트워 오버헤드가 코드 모듈과 시스템 간의 병렬 타입 일치로 얻어지는 이득을 상쇄하지 않도록 배려해야 된다. 이러한 스케줄링을 통하면 linear-speedup이 한계로 알려진 동기종 시스템과 달리 super-linear speedup을 달성할 수 있다[2].

최근에 유전자 알고리즘[3]이나 simulated annealing 알고리즘[4]과 같은 adaptive search 방법들이 그들이 갖는 범용성으로 인해 combinatorial optimization 문제들의 효과적인 해결 방법으로서 다양한 분야에 시도되고 있다 유전자 알고리즘은 population genetics에 기반을 둔 통계학적 서치 알고리즘이며 traveling salesman problem이나 classifier systems[5]과 같이 다양한 최적화 문제에 사용되고 있다. 본 논문에서는 다양한 내재 병렬 형태를 갖는 코드 모듈들로 구성된 프로그램을 분산 병렬 시스템에서 수행시키기 위하여 유전 자 알고리즘을 이용한 코드 모듈의 스케줄링 방법을 제안한다 본 논문의 구성은 다음과 같다. 2절에서는 문제 해결의 정형화를 위해 그래프를 이용한 프로그램과 분산 병렬 시스템의 표현 방법을 제시한다 제안 알고리즘의 구현 내용은 3절에서 기술되어지며 4절에서는 실험 결과를 통하여 제안 알고리즘을 분석하고 5장에서는 결론을 맺는다

Ⅱ. 문제의 정형화

본 논문에서 사용되는 분산 병렬 시스템은 다수의 고성능 및 병렬 컴퓨터 시스템들로 이루어진 point-to-point 네트워이다 각 시스템들은 특정 코드 형태의 프로그램의 수행에 대하여 보다 탁월한 효과를 보이는 특징을 갖는다 분산병렬 시스템은 다음과 같이 제안된 네트워 그래프로서 표현한다

정의 1 분산병렬 시스템 H 는 undirected 네트워 그래프 $G=\langle V_H, E_H\rangle$로 표현한다 V_H는 $\{ M_{i,k}, i \in \{1,2,\ldots,m\}, k \in \{1=SISD, 2=SIMD, 3=MIMD, 4=vector, \ldots\}\}$와 같이 정의되며 때로는 편의상, 시스템의 병렬 특성이 논점이 아닐 때 각 시스템들은 편의상 M_i와 같이 표현될 수도 있다 (integer k를 삭제) V_H 내의 각 시스템 $M_{i,k}$ 는 tuple - [k, R_{ik}]로서 특징지어지는데 k는 시스템 M 의 병렬 타입을 나타낸다 R_{ik} 는 분산병렬 시스템에 존재하는 타입 k의 시스템 중 가장 빠른 시스템과 비교한 시스템 $M_{i,k}$ 의 상대적 성능을 나타낸다 E_H 는 edges들의 집합이다 즉, M_i 가 링크를 통하여 M_j에 연결되어 있다면 하나의 edge를 형성하게 되므로 $((M_i, M_j) \in E_H)$, $E_H \subseteq V_H \times V_H$ 와 같이 쓸 수 있다 각 edge는 정수 값으로서 특징지어지며, 이 정수 값은 두 시스템간에 단위 정보를 전송하는데 소요되는 전송 경비를 나타낸다 즉, 이 정수 값은 transfer rate와 시스템들간의 타입 변환 오버헤드를 포함하는 대략의 추정 값이다. 이때 분산병렬 시스템의 각 시스템은 한번에 하나의 태스크만을 수행한다고 가정한다.

반면에, 병렬 프로그램은 다수의 코드 모듈 혹은 태스크들로 이루어지는데 이들 각 태스크들은 SISD, SIMD, MIMD 혹은 vector 타입과 같은 특성을 가지며 이들 태스크간에는 메시지의 전달을 통해 정보 교환이 이루어진다. 프로그램의 특성은 다음과 같이 태스크 그래프에 의해 표현된다

정의 2 프로그램 P는 directed, acyclic 태스크 그래프 $G_P=\langle V_P, E_P\rangle$에 의해 표시된다. $V_P=\{ t_{i,k}, i \in \{t, 1, 2, \ldots, n,\}, k \in \{1=SISD, 2=SIMD, 3=MIMD, 4=vector, \ldots\}\}$는 n 개의 태스크들로 구성되는 프로그램 P의 partition이며 이들 태스크들은 k 타입의 병렬특성을 갖는다. 편의상, 태스크의 병렬 특성이 논점이 아닐 때 태스크들은 t_i와 같이 표현될 수 있다 (integer k 삭제) V_P 를 구성하는 각 태스크 $t_{i,k}$는 tuple - [k,($D_{k,1}$, $D_{k,2}$, ., $D_{k,1}$), $O_{i,k}$]에 의해 특징지어지는데 $D_{k,1}$ 는 태스크 $t_{i,k}$ 와 시스템 $M_{j,1}$사이의 병렬 타입 불일치 시 감수해야 하는 성능 패널티를 나타내는 계수이다 가령 k = 1의 관계가 있다면 $D_{k,1}$ 는 1이란 값을 갖게 된다 1은 분산병렬 시스템에 존재하는 시스템들이 갖는 타입의 종류를 나타낸다. $O_{i,k}$ 는 $t_{i,k}$가 k 타입의 코드를 가장 잘 수행할 수 있는 시스템에서 수행되었을 경우 얻을 수 있는 최적 수행 시간이다 E_P 는 다음과 같이 directed edge들의 집합이다. 즉, 만일 데이터 의존성이나 두 태스크 간에 전달되어야 될 메시지의 필요성과 같은 이유로 인해 ti가 tj이전에 수행되어야 한다면 하나의 directed edge를 형성하게 되므로$((t_i, t_j) \in E_P)$, $E_P \subseteq V_P \times V_P$와 같이 쓸 수 있다 각 edge들은 두 태스크간에 전달되

어야 하는 정보의 양을 나타내는 integer 값으로 특징지어진다. 각 시스템에서의 태스크의 수행은 non-preemtive 방식으로 이루어진다고 가정한다. 즉, 하나의 태스크가 수행이 시작되면 프로그램의 종료 시까지 수행이 이루어지게 된다.

분산병렬 시스템에서의 스케쥴링 문제는 Schedule Length를 최소화하기 위해 프로그램 P 의 태스크를 분산병렬 시스템 H 의 시스템들에게 할당하는 매핑 함수 $\Pi(\Pi V_P \rightarrow V_H)$ 와 동일 시스템에 할당 되어진 태스크들의 수행 순서를 찾는 문제로 귀결된다 일단 매핑 정보와 동일 시스템에 할당되어진 태스크들의 수행순서를 찾게 되면 태스크 그래프는 스케쥴 태스크 그래프로 변환된다. Schedule Length는 스케쥴 태스크 그래프의 Critical Path 상에 존재하는 모든 태스크들의 수행시간과 통신 비용의 합으로 정의되어지며 이 때 동일 시스템에서 수행되는 두 태스크간의 통신 비용은 무시한다

Ⅲ. 유전자 알고리즘 기반의 태스크 스케쥴링 (GATS)

본 논문에서 제안하는 알고리즘은 그림1과 같이 구성되는데 초기 스트링 Population을 가지고 동작을 시작하여 여러 세대(Generation)를 거치면서 유전자 연산을 이용해 Population의 적합성(Fitness)을 개선시킨다. 이때 각 스트링은 적합치를 갖는다 제안 알고리즘은 두 개의 REPEAT 루프를 포함하는데 안쪽의 루프는 이전 세대에 속한 전체 스트링들 중 선택 정책(Selection)에 의해 선택되어진 두 개의 스트링들을 조합하거나 (Crossover 연산) 혹은 한 스트링의 일부를 변경 시키는 방법(Mutation 연산)에 의해 새로운 스트링들을 생성함 으로서 다음 세대를 구축하게 된다 즉, 안쪽 루프는 새로운 세대를 구성할 모든 스트링들이 만들어질 때까지 반복 수행되어지는데 루프가 1회 수행될 때 마다 새로운 스트링 한 개가 생성되며 한 세대를 구성하는 스트링 수가 루핑 횟수를 결정하게 된다 한 세대를 구성하는 스트링 개수는 임의의 숫자로 결정할 수 있다.

스트링 선택 시에는 지금까지 얻어진 스트링들 중 가장 좋은 적합치를 갖는 스트링이 다음 세대 안에 반드시 포함될 수 있도록 elitist 정책이 사용된다. elitist 정책은 Crossover 및 Mutation 연산이 갖는 분열 효과를 보완하는 효과가 있다 제안된 알고리즘에서는 한 세대를 구성하는 스트링 간의 경쟁 수준을 제한하기 위해 Scaling 정책도 사용된다. 반면에 밖의 루프는 종료 조건이 만족될 때까지 안쪽 루프가 반복 수행되도록 제어하며 종료 조건이 만족되는 경우 수행을 끝내게 된다. 종료 조건으로서는 여러 세대를 거치는 동안 적합치의 개선이 이루어지지 않는 경우 수렴한 것으로 판단하여 종료시키는 방법이 사용되었다

```
Generate the initial population - OldPop
Calculate the fitness of each schedule in OldPop
Find best_string among OldPop
prebest = best_string
Scale( OldPop )
generation = 0
REPEAT
  REPEAT
    string1= Select( OldPop ), string2 = Select( OldPop )
    Crossover_Mutation( string1, string2 )
  UNTIL all schedules are generated
  Elitist(prebest)
  Calculate the fitness of each string in NewPop
  Find best_string among NewPop
  prebest = best_string
  Copy( OldPop, NewPop )
  Scale( OldPop )
  generation++
UNTIL termination condition is met
```

그림 1 유전자 알고리즘 기반의 스케쥴링 개요

3 1 스케쥴의 표현과 적합성

Schedule Length의 계산은 스트링이 갖고 있는 스케쥴 정보를 이용하여 이루어진다. 이를 위해 본 논문에서는 스트링에 코딩될 스케쥴의 두 가지 표현 방법을 제안한다. 각 스트링은 일련의 병렬 및 벡터 시스템에 태스크들이 어떻게 할당되었는지를 표현한다 첫 번째 표현법에서는 스트링을 태스크 개수만큼의 세그먼트들로 나누고 각 세그먼트들은 특정 태스크가 고정적으로 사용하도록 한다 각 세그먼트에 코딩 되는 값은 특정 시스템의 식별자를 이진법으로 나타낸다. 이는 특정 세그먼트를 나타내는 태스크가 세그먼트에 코딩된 식별자가 나타내는 시스템에 할당되어졌음을 의미하게 된다 따라서 스트링을 구성하는 각 세그먼트는$\lceil \log_2 m \rceil$ bit들로 표현되게 되며 하나의 스트링은$\lceil \log_2 m \rceil *$ n bit들로 표현되게 된다 이때 m은 분산병렬 시스템에 존재하는 시스템의 숫자를, n은 태스크 그래프에 나타나는 태스크의 개수를 각각 나타낸다. 반면에 두 번째 방법에서는 하나의 스트링은 태스크의 할당 정보뿐 아니라 수행순서를 동시에 표현하는데 첫번째 방법과 달리 각 세그먼트는 특정 태스크에 고정적으로 사용되지 않는다 따라서 각 세그먼트는 특정 태스크

의 식별자와 이 태스크가 할당되는 시스템의 식별자를 동시에 갖게된다. 태스크 간의 수행순서는 스트링 안에서 어떤 세그먼트에 위치하느냐에 따라 결정된다 각 세그먼트는$\lceil \log_2 n \rceil + \lceil \log_2 m \rceil$ b$\lceil \log_2 n \rceil + \lceil \log_2 m \rceil$) * n bit들이 필요하게 된다 본 논문에서는 첫 번째 표현 방법에 대해서만 제안 알고리즘의 구현 방법을 기술한다

유전자 알고리즘은 특정 스케줄을 나타내는 스트링의 적합치를 최대화하는 과정을 반복하면서 최적의 스케줄 정보를 만들어낸다 더 높은 적합치를 갖는 스트링들은 다음 세대의 Population을 생성할 때 더 많은 기여를 할 가능성이 있다. 본 논문에서는 특정 스트링의 적합치를 다음과 같이 이 스트링이 나타내는 Schedule Length와 한 Population을 구성하는 모든 스트링들이 갖고 있는 Schedule Length 값들 중 최대 값과의 차이로 정의한다:

$$Fitness(s) = K - schedule\_length(s)$$
$$where\ K = \max_{s \in S} schedule\_length(s) \quad (1)$$

따라서 Schedule Length가 적은 스트링일수록 더 적합한 스트링으로 간주된다. 제안된 알고리즘의 임무는 적합성 함수에 대한 글로벌 최대치를 찾는 것이다 적합성 함수는 $F: S \rightarrow R^+$로 표현되며 이때 S는 스트링의 집합이며 R^+는 양의 실수로 표현되는 적합치를 나타내게 된다.

3 2 최초 스트링 Population의 생성 및 유전 연산자의 구현

일반적인 유전자 알고리즘은 최초 스트링들을 랜덤하게 생성하나 본 논문에서는 알고리즘의 수렴 시간을 향상시키기 위해 두 가지 초기화 방법을 제안한다. 첫 번째로 제한적 랜덤(Restricted Random) 초기화 방법에서는 스트링의 각 세그먼트마다 시스템의 식별자를 랜덤하게 할당하되 세크먼트가 나타내는 태스크의 병렬 타입과 어느 정도 부합되는 시스템 그룹들만을 대상으로 한다 이는 분산병렬 시스템에서는 부하 균등보다는 태스크와 시스템간의 병렬타입의 일치가 더욱 중요하기 때문이다. 따라서 병렬 타입의 일치가 고려되지 않은 스트링들은 제안 알고리즘의 수렴 시간을 연장시키게 된다. 두 번째로 리스트 (List) 초기화 방법에서는 리스트 스케줄링에[6] 의해 얻어진 스트링들을 초기 Population에 포함시킨다 리스트 스케줄링 역시 병렬 타입의 일치를 고려하여 설계되었다.

Population을 구성하는 스트링들은 Crossover나 Mutation과 같은 유전 연산에 의해 새로운 스트링들로 변환되어 새로운 Population을 형성한다 제안 알고리즘에서는 Crossover 연산이 최적화를 위한 주된 수단으로 사용되는데 현재 Population 중 선택 정책에 의해 선택된 두 스트링의 부 스트링을 혼합하여 새로운 스트링들을 생성하게 된다 이때 세그먼트가 부적절한 시스템 식별자 값을 갖지 않도록 유의해야 한다

Crossover 연산이 현재 존재하는 유전자에 대한 계승 메커니즘이라면 Mutation 연산은 스트링에 코딩 되어진 스케줄링 정보에 대한 간헐적인 랜덤 변경을 시도한다. 이는 계승 메커니즘으로는 소개되기 어려운 새로운 형태의 스케줄링 정보가 Population에 소개될 수 있도록 하여 스트링의 다양성을 증대시키기 위한 목적이 있다. 이는 서치 과정 중 알고리즘이 국부적인 최소점에 빠져 조기 수렴되는 것을 방지하게 된다. 본 논문에서는 4가지 유형의 Mutation 연산을 제안한다. 1) Global Swap (GS) - 무작위 선택된 두 태스크가 병렬 타입에 무관하게 할당 정보를 교환한다. 2) Global Migration (GM) - 무작위 선택된 태스크가 다른 시스템으로 할당되도록 할당 정보를 변경한다. 이 경우 새로이 할당될 시스템의 병렬 타입이 태스크의 타입과 일치할 필요는 없다 3) Restricted Global Migration (RGM) - 무작위 선택된 태스크가 다른 시스템으로 할당되도록 할당 정보를 변경하되 새로이 할당될 시스템의 병렬 타입과 태스크의 타입이 같거나 최소한 비슷하도록 고려한다 4) Hybrid - GM과 RGM 혹은 GS와 RGM의 혼합형이다 GM이나 GS Mutation은 타입 불일치를 허용하는 할당을 가능하게 하여 시스템간 부하 균형이 가능하도록 한다 이 때 GM 및 GS 연산의 빈도는 미리 정의된 확률에 의해 제어된다

Ⅳ. 실험 결과 및 분석

본 절에서는 3.2절에서 기술된 두 가지의 초기화 방법과 Mutation 방법들이 제안 알고리즘에 사용될 경우 제안 알고리즘이 나타내는 성능을 실험을 통하여 비교 분석한다 또한 제안 알고리즘과 일반적 리스트 스케줄링 방법 (HLFET)[7], 그리고 Simulated Annealing 기반의 태스크 스케줄링 방법과의 비교분석 내용을 기술한다. 실험에 사용된 제안 알고리즘은 다음과 같은 설정 값을 통하여 동작

시켰다. one-point crossover 및 80 % crossover rate 그리고 2 % mutation rate. 또한 Population의 크기는 태스크의 개수와 같게 하였다. 그리고 무작위 생성한 200개의 태스크 그래프와 20개의 네트웍 그래프를 대상으로 실험을 수행하였다.

우선 초기화 방법의 경우, 그림4에 제시된 바와 같이 제안된 알고리즘의 초기 Population을 구성하는 스트링 생성을 위해 제한적 랜덤 초기화 방법이나 리스트 초기화 방법을 사용하는 것이 단순한 랜덤 초기화 방법을 사용하는 경우에 비해 훨씬 효율적이다. 랜덤 초기화에 의해 생성된 초기 Population은 제한적 랜덤 초기화나 리스트 초기화 방법에 비해 낮은 평균 적합치를 갖는데 이는 다수의 태스크들이 병렬 타입이 일치하지 않는 시스템에 할당되는 경우가 많기 때문이다. 이 경우 우수한 유전자의 소개를 위해서 여러 세대를 거쳐야 되는 현상을 야기하게 되고 따라서 제안 알고리즘의 수렴이 늦게 이루어진다. 한편 리스트 초기화를 이용하는 경우 가장 우수한 결과를 나타낸다. 이것은 초기 Population에 이미 최적치에 근사한 스케쥴 정보가 포함되기 때문이다. 그러므로 이 경우 초기 스케쥴에 비해 괄목할만한 개선이 이루어지지는 않으나 개선과 동시에 알고리즘의 수행이 상대적으로 빨리 종료되는 장점이 있다. 그러나 이 경우 알고리즘의 조기 수렴을 방지하기 위해서는 Mutation연산이 적절히 사용되어야 된다.

Mutation 연산의 경우, 그림 5에 제시된 바와 같이 RGM 연산은 GS나 GM 연산에 비해 우수한 성능을 나타낸다. 유전자 알고리즘에서 Mutation 연산은 Crossover 연산이 만들어내지 못하는 중요한 유전자의 소개를 가능케 하여 스트링의 다양성을 제공하는 목적을 갖는다. 그러나 분산병렬 시스템에서 더 많은 성능 향상을 얻기 위해서는 대부분의 태스크가 타입이 일치하는 시스템으로 할당이 이루어져야 가능하다. 그러므로 분산병렬 시스템에서 랜덤 다양성은 할당 정보가 반드시 병렬 타입에 무관하게 변화되어야 함을 의미하지는 않는다. 이는 RGM 연산을 채택한 알고리즘이 더 우수한 스케쥴 정보를 만들어 내면서도 더 빨리 수렴한다는 점에서도 알 수 있다. GS 혹은 GM 연산은 RGM 연산에 비해 상대적으로 분열성의 특징을 나타내며 이는 조기 수렴을 어렵게 하는 결과로 나타난다.

한편, 분산병렬 시스템에서 병렬 타입의 일치가 더 많은 속도 향상의 원동력이기는 하지만 병렬 타입의 일치가 반드시 최적 수행 시간을 보장하지 않을 수도 있다. 즉, 일부 태스크들을 병렬 타입이 불일치하는 시스템에 할당함으로서 Schedule Length가 감소될 수 있다. 예를 들어 타입이 일치하는 시스템으로의 이동 시 통신 오버헤드가 과중한 경우나 시스템간의 이질성이 그리 크지 않을 때 혹은 시스템간의 부하가 심한 불균형을 이루는 경우가 해당될 수 있다. 본 논문에서는 이 문제의 해결을 위해 Hybrid Mutation 연산을 사용한다. Hybrid Mutation 연산은 RGM과 GM의 혼합형으로, 대개의 경우 RGM연산을 통해 태스크와 시스템간의 타입이 일치하는 범위 내에서만 할당 정보 변경을 허락하나 간헐적으로 GM, GS 연산을 통하여 타입 불일치 할당도 가능하게 한다. 이 경우 그림 3과 같이, Hybrid Mutation을 채택한 제안 알고리즘이 RGM을 채택한 경우보다 GM이 갖고 있는 분열 성격으로 인해 수렴은 다소 늦게 이루어지나 보다 우수한 스케쥴을 만들어낸다.

표 1,2,3은 제안 알고리즘과 Simulated annealing을 이용한 방법간의 효율성을 비교한다. 그림 2에 기술된 후자의 경우에도 초기 스트링의 생성과 Mutation 연산에 상응하는 Perturbation 연산의 구현에 제안 알고리즘과 같은 원칙을 적용하였다. 즉, 초기 스트링의 생성시 제한적 랜덤 초기화 방법이나 리스트 초기화 방법을 사용하였고, get_perturbed_schedule(S)에서도 RGM 연산을 작용하였다. 실험결과는 표 1,2,3에 보인 것과 같이 두 가지 방법 모두 그림3에 보인 HLFET에 비해 괄목할 만한 성능을 보였으나 제안 알고리즘이 수렴 시간 측면에서 보다 우수함을 보였다. Greedy 알고리즘의 성격을 갖는 HLFET 알고리즘은 사용 가능한 프로세싱 노드가 있으면 노드와 태스크간 병렬 타입의 고려 없이 바로 태스크의 할당을 하기 때문에 열악한 성능을 나타내게 된다.

표의 내용 중 Mean은 한 세대를 구성하는 스트링들이 표현하는 schedule length가 HLFET에 의해 얻어진 schedule length에 비해 평균적으로 어느 정도 우수한 해답을 생성하는지를 나타내며, Min은 수렴시 최종 세대를 구성하는 스트링들 중 가장 낮은 적합치를 갖는 스트링이 HLFET에 비해 어느 정도 우수한 해답을 생성하는지를 나타낸다. Max는 수렴 시 최종 세대를 구성하는 스트링들 중 가장 높은 적합치를 갖는 스트링이 HLFET에 비해 어느 정도 우수한 해답을 생성하는지를 나타낸다. 반면에 Median은 최종 세대를 구성하는 스트링들 중 중간 정도의 적합치를 갖는 스트링이 HLFET에

비해 어느 정도 우수한 해답을 생성하는지를 나타낸다 표준편차는 최종 세대에 속한 스트링들이 갖는 적합치의 분포 상황을 나타내고 있다 실험은 무작위 생성한 200개의 태스크 그래프와 20개의 네트워 그래프를 대상으로 수행되었는데, 태스크 그래프의 경우 태스크 개수(N)를 4 - 32 범위 안에서 생성하였고, 네트워 그래프의 경우 노드의 개수(M)를 4 - 8의 범위 안에서 생성하였다. 표의 데이터는 특정 (N,M)의 경우 각 알고리즘 별로 10회 반복 수행하여 얻은 결과를 평균하여 표시하였다

```
set an initial temperature - T
get an initial schedule - S_cur
calculate schedule length of the initial schedule - C_cur
min ← C_cur
REPEAT
  FOR each segment in a string
   S_new ← get_perturbed_schedule( S_cur )
   calculate schedule length of new schedule - C_new
   ΔC = C_new - C_cur
   IF ( ΔC <= 0 ) S_cur ← S_new
   ELSE S_cur ← S_new with probability of exp (- ΔC/T)
   ENDIF
   IF ( C_new < min) min ← C_new
   ENDIF
  ENDFOR
  T ← αT
UNTIL termination condition is met
IF ( min< C_cur) return( min )
ELSE return( C_cur)
ENDIF
```

그림 2 Simulated Annealing 기반의 스케줄링 개요

```
Assign a priority to each task in task graph
build a priority_list
build initial ready_list with tasks having no
predecessors
REPEAT
  IF any available machine exists
    REPEAT
      select a first ready task in priority_list
      assign selected task to an available machine
      delete selected task from priority_list, ready_list
      enter the children of selected task to ready_list
    UNTIL all available machines are assigned task
  END IF
UNTIL ready list is empty
```

그림 3 HLFET 개요

V. 결 론

본 논문에서는 다양한 내재 병렬 형태를 갖는 프로그램의 수행을 고속 네트워 기반의 분산 병렬 시스템을 통하여 수행하기 위한 유전자 알고리즘 기반의 태스크 스케줄링 방법을 제안하였다 이때 더 많은 속도향상을 얻기 위해서는 부하균형과 아울러 각 태스크들이 시스템에 할당될 때 태스크의 코드 타입과 시스템의 병렬특성을 되도록 일치시켜야 되며 동시에 태스크의 이동으로 인한 통신 오버헤드가 최소화 되어야 한다 분산병렬 시스템에서는 부하 균형보다는 오히려 병렬 타입의 일치가 성능 향상의 주된 요인이다 제안된 유전자 알고리즘 기반의 스케줄링에서는 분산병렬 시스템이 갖는 특징을 이용하여 수렴 시간을 단축시킬 수 있다. 본 논문에서는 두 가지의 초기화 방법과 Mutation 방법을 제안하였으며 실험 결과를 통하여 제안된 방법이 효과적인 결과를 만들어냄을 보였다 또한 Simulated Annealing을 이용한 스케줄링 방식과 비교하여 제안 알고리즘이 보다 효율적임을 보였다

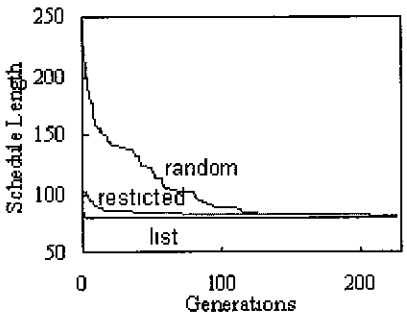

그림 4 초기화 방법에 따른 알고리즘 동작

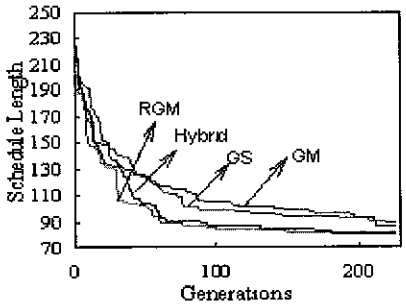

그림 5 Mutation 방법에 따른 알고리즘 동작

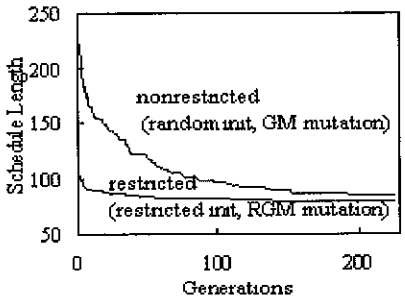

그림 6 스트링 공간에 따른 알고리즘 동작

표 1. 각 알고리즘의 HLFET 대비 성능 개선(N=16, M=6)

	Mean	min	max	media-n	표준편차	CPU time
GATS	66.2%	61.1%	72.1%	65.5%	4.8%	≈4.5 sec
GATS (list init)	68.4%	63.1%	76.8%	66.3%	5.6%	≈1.2 sec
SATS	66.0%	61.2%	71.3%	65.3%	4.9%	≈5.3 sec

표 2. 각 알고리즘의 HLFET 대비 성능 개선(N=24, M=6)

	Mean	min	max	media-n	표준편차	CPU time
GATS	63.9%	57.2%	67.6%	65.0%	4.1%	≈10.8 sec
GATS (list init)	65.6%	59.4%	69.2%	66.2%	4.0%	≈2.3 sec
SATS	64.5%	58.2%	69.3%	64.1%	4.4%	≈14.1 sec

표 3. 각 알고리즘의 HLFET 대비 성능 개선(N=32, M=6)

	Mean	min	max	media-n	표준편차	CPU time
GATS	61.7%	55.1%	67.8%	62.2%	6.4%	≈17.1 sec
GATS (list init)	63.6%	57.2%	68.9%	64.7%	5.9%	≈3.4 sec
SATS	61.3%	55.7%	67.8%	60.4%	6.1%	≈22.5 sec

참 고 문 헌

[1] H. S Kim, "Web/Java 기반의 고성능 분산 컴퓨팅 패러다임", 한국전자통신연구원 주간기술동향 포커스 논문,TIS-97-42 820호, 1997, pp. 16-31

[2] H.S. Kim, "고속 네트웍 기반의 분산병렬 시스템에서의 성능 향상 분석 모델", 한국통신학회 논문지 제26권 제 12호C, Dec. 2001, pp.218-224

[3] Mitsuo Gen, Genetic Algorithms & Eng. Optimization: John Wiley & Sons, 2000

[4] Van Laarhoven and E.H. Aarts, Simulated Annealing: Theory and Applications, The Netherlands, 1988

[5] G. Robertson, "Parallel Implementation of Genetic Algorithms in a Classifier System", Genetic Algorithms and Simulated Annealing, edited by L. Davis, Morgan Kaufmann, 1987, pp. 129-140

[6] H.S Kim, S.M Kang, "List Scheduling in High Speed Network based Distributed Parallel Systems", ICACT 2000, Feb. 2000, pp. 491-496

[7] T.L. Adam, K.M. Chandy and J.R.Dickinson," Comparison of List Schedules for Parallel Processing Systems" Communications of the ACM, Dec. 1974, pp. 685-690

김 화 성(Hwa-sung Kim) 정회원

1981년 2월 : 고려대학교 전자공학과 학사

1983년 2월 : 고려대학교 전자공학과 석사

1996년 10월 : Lehigh Univ. 전산학과 박사

1984년 3월~2000년 2월 : ETRI 책임 연구원

2000년 3월~현재 : 광운대학교 전자공학부 교수

<주관심 분야> 차세대 네트워크 구조, 미들웨어 환경, 그리드 컴퓨팅, 이동환경에서의 QoS