논문 2004-41CI-2-2

2차원 토러스 기반 다중 디스크 데이터 배치 병렬 유전자 알고리즘

(A 2-Dimension Torus-based Genetic Algorithm for Multi-disk Data Allocation)

안 대 영*, 이 상 화**, 송 해 상**

(Dae-Young Ahn, Sang-Wha Lee, and Hae-Sang Song)

요 약

본 논문에서는 NP-Complete 부류에 속하는 다중 디스크 데이터 배치 문제를 해결하기 위한 병렬 유전자 알고리즘을 제안한다. 이 문제는 디스크 입출력 처리의 병렬성이 극대화되도록 Binary Cartesian Product File의 데이터 블록들을 디스크어레이에 배치하는 방식을 찾는 것이다. 이 문제를 해결하기 위하여 제안되었던 DAGA 방식은 순차 유전자 알고리즘(Genetic Algorithm)으로서, 이전에 제안되었던 다른 방식에 비해 디스크 수에 대한 제약을 없애면서도 우수한 결과를 제공함을 보여주었으나 시뮬레이션 시간이 너무 커서 큰 용량의 데이터 구성에 대한 시뮬레이션을 어렵게 하는 문제점이 있었다. 본 논문에서는 DAGA의 시뮬레이션 시간 단축을 위한 방식으로서, 2차원 토러스(2-Dimension Torus) 기반 병렬 유전자 알고리즘(ParaDAGA)을 제안한다. ParaDAGA는 분산 객체 모형을 기반으로 설계되었으며, 단일 프로세서 시스템에서 구현된 병렬처리 컴퓨터 시뮬레이터에서 수행되도록 구현하였다. 시뮬레이션 연구를 통하여, ParaDAGA의 시뮬레이션 변수 값이 결과에 주는 영향을 분석하였고, ParaDAGA 방식이 DAGA 방식에 비해 우수한 결과를 제공할 수 있는지를 실험하였다. 실험 결과는 ParaDAGA 방식이 순차 알고리즘인 DAGA보다 알고리즘 수행 시간 뿐 아니라, 찾아낸 결과도 우수함을 보여준다.

Abstract

This paper presents a parallel genetic algorithm for the Multi-disk data allocation problem, an NP-complete problem. This problem is to find a method to distribute a Binary Cartesian Product File on disk-arrays to maximize parallel disk I/O accesses. A Sequential Genetic Algorithm(SGA), DAGA, has been proposed and showed the superiority to the other proposed methods, but it has been observed that DAGA consumes considerably lengthy simulation time. In this paper, a parallel version of DAGA(ParaDAGA) is proposed. The ParaDAGA is a 2-dimension torus-based Parallel Genetic Algorithm(PGA) and it is based on a distributed population structure. The ParaDAGA has been implemented on the parallel computer simulated on a single processor platform. Through the simulation, we study the impact of varying ParaDAGA parameters and compare the quality of solution derived by ParaDAGA and DAGA. Comparing the quality of solutions, ParaDAGA is superior to DAGA in all cases of configurations in less simulation time.

KeyWords : disk allocation, genetic algorithm, parallel I/O

* 정회원, 서원대학교 컴퓨터교육과
(Department of Computer Education, Seowon University)

** 정회원, 서원대학교 컴퓨터정보·통신공학부
(Department of Computer Information & Communication, Seowon University)
접수일자 : 2003년 9월 30일, 수정완료일 : 2004년 2월 26일

I. 서 론

컴퓨터와 네트워크의 발달에 기인한 인터넷 기반 컴퓨팅 환경의 보편화로 인하여, 데이터의 사용이 폭발적으로 증가하고 있다. 현재 사용되고 있는 컴퓨터는 고속의 데이터 저장 및 추출을 위하여 하드 디스크 장치를 주 저장 매체로 사용하고 있으며, 이에 따라 디스크 입출력의 성능이 전체 시스템 성능을 결정하는 중요한 요소가 되었다. 병렬 디스크 입출력 시스템인 디스크 어레이는 디스크 입출력 성능 향상을 위해 제안된 방식으로서, 디스크 입출력 요구가 여러 디스크에서 병렬로 처리되게 함으로써 입출력 응답시간을 줄인다. 그러나 저장될 데이터를 다중 디스크에 적절히 분산시키지 않으면, 특정 디스크에 입출력 요구가 편중되어 병렬성이 감소되기 때문에 예상했던 성능 증가를 보장받을 수 없다. 본 논문에서 다루는 다중 디스크 데이터 배치 방식은, 다중 디스크에서의 데이터 입출력의 병렬성이 극대화되도록 데이터들을 다중 디스크에 최적으로 분산 배치하는 방법을 찾는 것이다. 최적의 데이터 배치 방식을 찾는 문제는 NP-complete[3] 부류에 속하는 문제로 알려져 있기 때문에, 기존의 제안된 대부분의 방식들은 근사 최적 방식을 찾기 위해 휴리스틱(heuristic) 접근 방식을 이용하고 있다.

최근에 제안된 방법 중의 하나인 DAGA(Disk Allocation method based on Genetic Algorithm)[1]는 다중 어트리뷰트 파일(Multiple attribute file) 중의 하나인 Binary Cartesian Product file(BCPF)을 다중 디스크에 배치하는 방식으로서, 순차적 유전자 알고리즘(Sequential Genetic Algorithm)[2][3]을 기반으로 설계되었다. 유전자 알고리즘은 생태계의 유전자 교환을 통한 진화 방식과 적자생존 법칙을 모형화한 탐색(search) 방식으로서, 다양한 분야의 최적화 분야에 성공적으로 적용되었다. DAGA의 실험 결과는 기존에 제안된 방식들에 비하여 다중 디스크의 수에 대한 제한을 제거하면서도 우수한 데이터 배치 결과를 제공할 수 있음을 보여준다. 그러나 DAGA에서는 각 데이터 배치 결과의 우수성을 판별하는 값으로서, 모든 조합 가능한 입출력 요구에 대한 평균 디스크 처리 시간을 이용하기 때문에 알고리즘의 시뮬레이션 수행 시간이 매우 크다는 단점이 있다. 비록 DAGA에 의한 데이터 배치는 다중 어트리뷰트 파일을 다중 디스크에 만들 때에 한번만 수행되는 오프라인(off-line) 알고리즘이지만, 긴 시뮬레이션 시간은 큰 구성의 파일 구성을 위해 DAGA 알고리즘을 수행시킬 때 시뮬레이션 시간이 너무 커서 큰 용량의 데이터 구성에 대한 시뮬레이션을 어렵게 하는 문제점이 있다

유전자 알고리즘의 수행시간을 줄이는 방법은 순차 유전자알고리즘을 병렬처리 컴퓨터에서 수행되도록 병렬화 하는 것으로서, 병렬 유전자 알고리즘(Parallel Genetic Algorithm)과 관련된 많은 연구들이[5]-[11] 이러한 노력의 일환으로 수행되었다. 대부분의 병렬 유전자 알고리즘은 분산 객체 구조(distributed population structure)를 기반으로 구현되었는데, 이 구조에서는 순차 유전자 알고리즘에서 적용되었던 객체군(Population)이 여러 개의 서브 객체군(Sub-populations) 으로 나누어지고, 각 서브 객체군은 병렬처리 컴퓨터 시스템의 각 프로세서 노드에 하나씩 할당되어 독립적으로 동시에 진화(evolution)한다. 즉, 모든 노드는 서로 다른 데이터에 대하여 같은 유전자 알고리즘 프로그램을 수행하는 데이터 병렬 계산 모델(data parallel computational model)을 기반으로 구현되며, 주기적으로 다른 노드들과 필요한 정보를 교환한다. 이러한 구조를 가지는 병렬 유전자 알고리즘은 시뮬레이션 수행 시간 면에서 노드의 수에 따라 선형적인 성능 증가를 제공하는 것으로 알려져 있다[5][6].

본 논문에서는 다중 디스크 데이터 배치 문제를 해결하기 위해 제안되었던 순차 유전자 알고리즘 DAGA[1]의 시뮬레이션 시간 단축을 위한 방식으로서, 2차원 토러스(2-Dimension Torus) 기반 병렬 유전자 알고리즘 ParaDAGA(Parallel implementation of DAGA)을 제안한다. ParaDAGA는 분산 객체 구조를 기반으로 설계되었으며, 단일 프로세서 시스템에서 구현된 병렬처리 컴퓨터 시뮬레이터에서 수행되도록 구현하였다. 병렬 유전자 알고리즘에 관한 연구들로부터 선형적인 시뮬레이션 수행시간 개선은 많이 알려져 있기 때문에, 본 논문에서는 ParaDAGA의 시뮬레이션 변수 값이 결과에 주는 영향과 ParaDAGA 방식이 DAGA 방식에 비하여 우수한 결과를 제공할 수 있는지의 여부를 알기위한 실험을 중점적으로 수행하였으며 결과를 제시하였다.

본 논문의 구성은 다음과 같다. II에서는 다중 디스크 데이터 배치 문제를 정의하고, III장에서 병렬 유전자 알고리즘인 ParaDAGA의 알고리즘과 실험을 위한 시뮬레이터의 구현 내용을 기술한다. IV장에서는 실험 결과를 통하여 DAGA와 ParaDAGA의 결과를 비교하고, ParaDAGA의 시뮬레이션 변수 값의 변화에 대한 결과를 분

석한다. 끝으로 V장에서는 결론을 제시한다.

II. 문제 정의

본 논문에서 다루는 다중 디스크 데이터 배치 문제는 디스크들로부터의 입출력 처리가 최대로 병렬로 이루어지도록 Binary Cartesian Product File(BCPF)을 디스크 어레이에 배치하는 방식을 찾는 문제이다. BCPF는 n개의 어트리브트를 가진 버킷(bucket)들로 구성된 파일로서, 각 버킷은 $[i_1, i_2, \ldots, i_n]$ 로 표현되고 각 구성요소($i_1, i_2, \ldots, i_n$)의 값은 0과 1의 값을 가진다. 그림 1은 m개의 디스크에 4개의 어트리뷰트를 가지는 BCPF가 저장된 한 예를 보여준다. BCPF에 대한 입출력 요구는 부분 일치 질의어(partial match queries)의 형태로 시스템에 주어지는데, 부분일치 질의어는 각 어트리뷰트의 값이 0또는 1로 주어지거나 미정의 값(*로 표현)으로 표현된다. 부분 일치 질의어가 실제로 입출력을 요구하는 버킷들은 미정의 어트리뷰트에 표시될 수 있는 모든 조합의 어트리뷰트 값에 해당하는 버킷들이다. 예로서, 그림 1의 부분일치 질의어 [01**]에 응답되는 질의 응답 집합(query response set)은 {[0100], [0101], [0110], [0111]} 등 4개의 버킷 요소이며 , 이들은 m개의 디스크들에 분산되어 저장된 BCPF에서 읽혀지게 된다.

주어진 부분 일치 질의어에 대한 질의 응답시간은 $\max\{N_0, N_1, \ldots, N_{m-1}\}$로 정의되며, 여기서 $N_i(0 \le i \le m-1)$는 디스크 i 에서 응답하는 버킷의 수이다. 그림 1의 경우 버킷 [0100]과 [0101]이 디스크 d_{m-1}에 위치해 있기 때문에 질의 응답시간은 2가 된다. 만약 두개의 버킷 [0100]과 [0101] 중의 하나가 디스크 $d_k(0 \le k \le m-2)$에 배치되었다면 질의 응답시간은 1이 된다. 따라서 다중 디스크 데이터 배치 문제는 n 어트리뷰트를 가지는 버킷들로 구성된 BCPF와 m 개의 디스크가 주어졌을 때, 부분 일치 질의어에 대한 질의 응답시간을 최소화하도록 BCPF의 버킷 집합 $B=\{d_i \mid 0 \le i \le 2^n-1\}$를 m개의 디스크에 배치하는 방법을 찾는 문제로 정의된다.

DAGA에서는 버킷벡터(bucketvector)와 디스크 사상함수(disk mapping function)을 정의하여 다중 디스크 데이터 배치 문제를 버킷벡터 영역에서 가장 우수한 버킷 벡터를 찾아내는 탐색(search)문제로 변환하고, 탐색 방식으로서 유전자 알고리즘을 적용하였다. DAGA의 설계 개념에서는 그림 2에서 보는 바와 같이 BCPF를 구성하는 모든 버킷들을 담을 수 있는 가상의 버킷 상자(bucket vessel)를 정의한다. 버킷 상자에는 한 개의 버킷을 담을 수 있는 방들(V_{ij})이 순서적으로 배열되어 있으며, 한 버킷 상자 내에서는 같은 버킷이 2개 이상 존재하지 않는다. 따라서 BCPF의 버킷들을 임의의 순서로 하나씩 선정하고, 선정된 버킷 $[i_1, i_2, \ldots, i_n]$의 이진 어트리뷰트 값 $i_1 i_2 \ldots i_n$ 에 대응하는 10진 값(예로써, 버킷 0100, 0111 은 각각 10진 값 4와 7에 대응)을 버킷 상자내의 각 방에 왼쪽부터 순서대로 저장하면 하나의 정수문자열이 만들어지는데, 이 문자열을 버킷벡터라고 정의한다. 다음으로, 정의된 버킷벡터에 대하여 다음과 같은 디스크 사상함수 $f: B \rightarrow D$ 를 정의 한다.

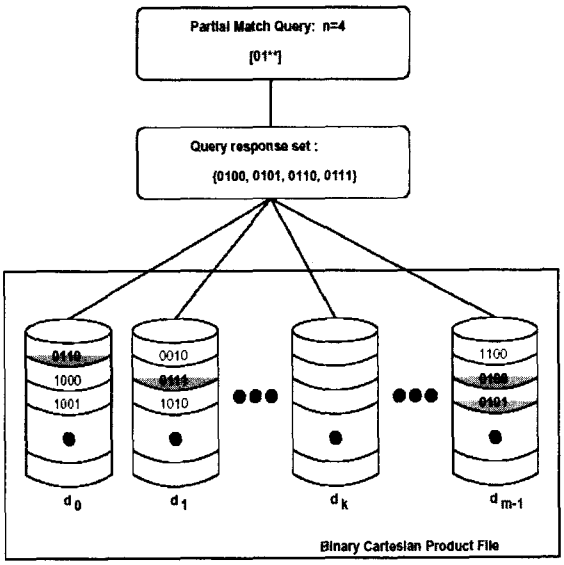


그림 1. 다중 디스크 데이터 배치 문제 : 예제.
Fig. 1. Multi-disk Data Allocation Problem: Example.

$$f(b_i) = j \bmod m \tag{1}$$

여기에서 집합 B는 버킷 벡터의 구성요소인 정수들의 집합이고 D는 디스크의 집합이며, j 는 버킷상자(버킷벡터)에서 버킷 b_i 가 들어있는 방의 위치 값이다.

그림 2에서 어트리뷰트의 수가 n이고, 디스크의 수가 m일 때, 디스크 사상함수 f 를 적용하면, 버킷방 $V_{k1}, V_{k2}, \ldots, V_{km}$에 저장된 버킷들은 디스크 $d_0, d_1, \ldots, d_{m-1}$ 로 각각 사상되어 저장된다.

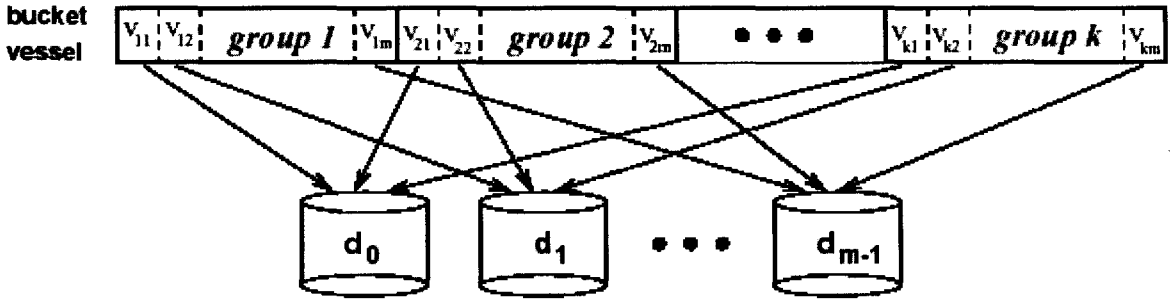


그림 2. DAGA의 설계 개념
Fig. 2. Design Concept of the DAGA.

위치 값	0	1	2	3	4	5	6	7	8
버킷벡터	2	4	6	1	8	0	3	7	5

그림 3. 버킷벡터 예 1.
Fig. 3. Bucketvector example 1.

위치 값	0	1	2	3	4	5	6	7	8
버킷벡터	1	3	7	0	6	4	2	5	8

그림 4. 버킷벡터 예 2.
Fig. 4. Bucketvector example 2.

예로서, 디스크의 수 $m=3$ 이고 어트리뷰트의 수 $n=3$ 일때 그림 3과 같은 순열을 가지는 버킷벡터는 디스크 사상함수를 적용했을 때 다음과 같은 버킷 배치 결과를 제공한다.

- 디스크 0에 배치되는 버킷 : 1, 0, 2
- 디스크 1에 배치되는 버킷 : 3, 6, 5
- 디스크 2에 배치되는 버킷 : 7, 4, 8

반면에 그림 4와 같은 버킷벡터의 경우는 다음과 같이 버킷들이 디스크들에 배치된다.

- 디스크 0에 배치되는 버킷 : 2, 1, 3
- 디스크 1에 배치되는 버킷 : 4, 7, 8
- 디스크 2에 배치되는 버킷 : 0, 5, 6

앞의 예에서 보는 바와 같이 서로 다른 정수 순열의 버킷 벡터는 디스크 사상함수의 적용에 의해 서로 다른 디스크 배치의 결과를 가져온다. 즉 버킷 벡터와 디스크 사상 함수를 도입하면 다중 디스크 데이터 배치문제는 다음과 같이 변환된다.

주어진 조건: 어트리뷰트 수 = n, 디스크의 수 = m
구하고자 하는 답: 다음의 정수 집합 B의 요소들로 구성되는 정수 순열들 중에서 다음의 목표를 만족시키는 하나의 순열

$$B = \{b_i \mid 0 \le i \le m \cdot \ 2^n./m \ -1,\ 0 \le b_i \le 2^n-1 \text{ and } b_j \neq b_k \text{ for } j \neq k\}$$

목표: 정수 순열의 판별 값(fitness value)을 최대화

DAGA에서 사용된 된 판별 값은 평균 질의 응답시간이며 다음과 같이 정의된다.

$$T_{avg} = \sum_{i=1}^{n-1} T_i \cdot P_i \qquad (2)$$

식 (2)에서 T_i 는 i개의 어트리뷰트가 미정의 값을 가진 모든 부분일치 질의어에 대한 응답시간을 평균한 값이고, P_i 는 모든 발생 가능한 부분일치 질의어 수에 대한 i 개의 미정의 어트리뷰트 값을 가지는 부분일치 질의어들의 비율이다. 따라서 T_{avg} 는 다음과 같이 표현된다.

$$T_{avg} = \sum_{i=1}^{n-1}\left(\frac{1}{C_i}\sum_{j=0}^{C_i}\max_j\{N_0, N_1, \ldots, N_{m-1}\}\right) \cdot \left(C_i \ / \ \sum_{k=1}^{n-1} C_k\right),$$

여기서 $C_i = \binom{n}{i}$ (3)

즉, DAGA에서는 다중 디스크 데이터 배치 문제를 여러 발생 가능한 정수 순열들 중에서 최적의 판별값에

근사하는 하나의 순열을 찾는 검색문제로 변환하고, 검색 문제를 풀기위한 휴리스틱 알고리즘으로서 유전자 알고리즘을 이용하였다.

Ⅲ. 2-D 토러스 기반 병렬 유전자 알고리즘(ParaDAGA)

순차 유전자 알고리즘 DAGA의 구성 및 세부 알고리즘은 참고 문헌[1]에 제시되었으며, 실험 결과는 기존에 제안된 방식에 비하여 다중 디스크의 수에 대한 제한을 제거하면서도 우수한 데이터 배치 결과를 제공할 수 있음을 보였다. 그러나 DAGA는 각 객체의 판별값을 계산하는데 상당히 많은 CPU 시간을 소비하는 것이 관찰되었다. 더 큰 n, m 의 구성에 대한 시뮬레이션 결과를 얻기 위해서는 객체군의 크기를 크게 하여야 하며, 이는 필연적으로 알고리즘 수행 시간이 크게 증가함을 의미하게 된다.

이러한 수행 시간의 증가 문제는 분산 객체군 구조를 가지는 병렬 유전자 알고리즘을 이용하면 해결될 수 있으며, 선형적인 수행 시간 향상을 얻을 수 있음을 많은 연구들이 보여준다. 본 논문에서 제안하는 유전자 알고리즘인 ParaDAGA는 DAGA가 2차원 토러스 토폴로지의 상호 연결망을 갖는 병렬 컴퓨터에서 수행 될 수 있도록 수정된 병렬 유전자 알고리즘이며, 분산 객체군(distributed population) 구조[11]를 가진다.

분산 객체군 모형은 그림 5 에서 보는 바와 같이 전체 객체군이 여러 개의 서브 객체군으로 나누진 후, 각 서브 객체군 별로 독립적으로 유전자 알고리즘에 의해 객체를 진화시킨다. 일정 기간(generation)의 독립적인 객체 진후 후에, 지형적으로 인접한 서브 객체군들은 주기적으로 서로 자신들에 속해 있던 객체들의 일부분을 인접한 상대 서브 객체군으로 이주(migration) 시킨다. 이러한 구조는 병렬 처리 시스템에 구현되기 적합하며, 순차 유전자 알고리즘에 비해 시뮬레이션 수행 시간을 크게 개선시킬 수 있을 뿐 아니라, 지역 최소치(local minima)에 빠지지 않고, 보다 양질의 결과를 제공해 줄 수 있다. 서브 객체군 간의 이주 객체수와 이주 주기(migration interval)는 병렬 유전자 알고리즘의 수행에 영향을 주는 중요한 요소로서, 이주 객체 수와 이주 주기의 변화에 대한 ParaDAGA의 특성은 Ⅳ장에서 실험적으로 고찰된다.

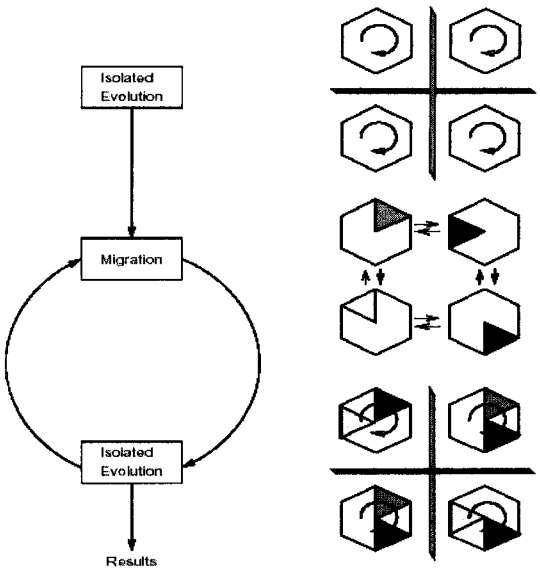


그림 5. 분산 객체군 구조 기반 병렬 유전자 알고리즘
Fig. 5. Parallel Genetic Algorithm based on distributed population structure.

1. 2차원 토러스 기반 병렬처리 컴퓨터의 시뮬레이터 구조

ParaDAGA가 수행될 2차원 토러스 기반 병렬 컴퓨터는 단일 프로세서 컴퓨터에서 시뮬레이터로 구현되었으며, 그림 6은 구현된 시뮬레이터의 세부 모습과 수행 환경을 보여 준다.

그림 6 에서 보는 바와 같이 병렬처리 컴퓨터의 각 프로세서 노드는 UNIX 프로세스로, 프로세서 노드간의 상호 연결망은 UNIX 프로세스간의 통신 API인 소켓(socket)으로 구현되었으며, 각 노드 프로세스는 4개의 이웃 노드와 연결되는 2차원 토러스 토폴로지를 구현하기 위하여 4개의 소켓 채널을 가진다.

ParaDAGA는 호스트(host) 프로그램과 노드(node) 프로그램으로 구성된다. 각 서브객체군은 노드 프로세스(node process)라는 UNIX 프로세스에 할당되고, 각 프로세스는 할당된 서브 객체군에 대하여 node() 알고리즘을 수행한다. 시뮬레이션 수행 초기에 하나의 호스트 프로세스(host process)만이 생성되어 host() 알고리즘의 수행을 시작하고, 노드 프로세스는 호스트 프로세스에 의해서 생성(creation)된다. 호스트 프로세스는 노드 프로세스의 생성 및 수행 제어 뿐 아니라, 시뮬레이션 결과를 알려주는 역할을 담당한다. INPUT 파일은 호스트 프로세스에게 전달될 시뮬레이션 정보를 담고 있으며, 노드 프로세스에서 사용될 여러 가지 변수들이 정의되어 있다. 호스트 프로세스는 주어진 정보에 따라

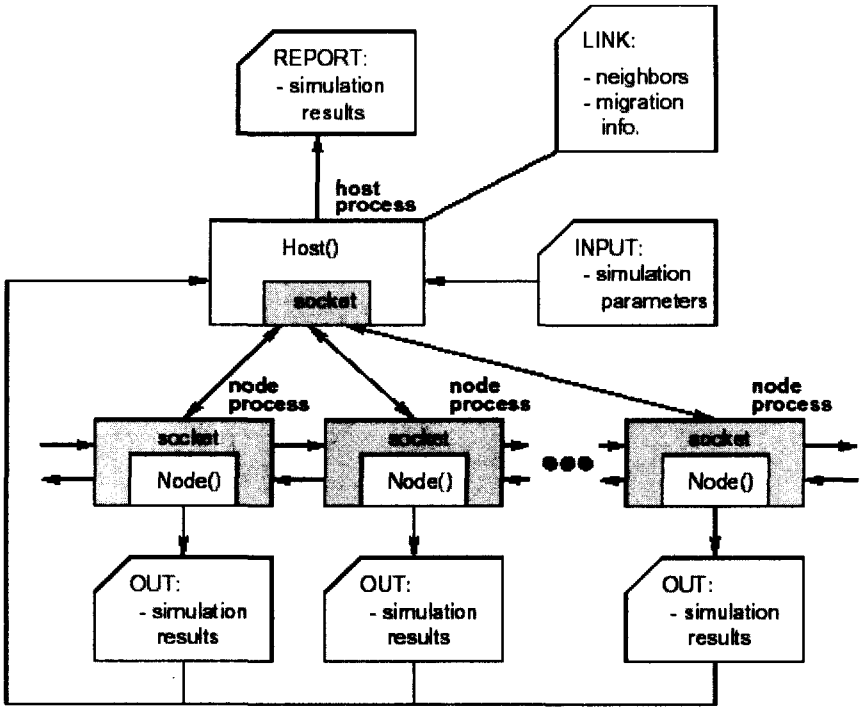


그림 6. 시뮬레이터 세부 모습 및 수행 환경
Fig. 6. Implementation Details of Simulator & Execution Environment.

이웃 정보(neighbouring information)과 이주 정보(migration information)을 만들며, 이 정보는 LINK 파일에 저장된다. 본 논문에서는 각 프로세스가 2차원 토러스 토폴로지로 연결된 경우를 가정하였다. 따라서 각 노드 프로세스는 정확히 4개의 이웃 노드 프로세스와 연결된다. LINK 파일에는 각 노드 프로세스 마다 이웃 관계에 있는 프로세스와 소켓 정보를 저장하고 있다. 각 노드 프로세스의 알고리즘 수행 결과는 OUT 파일에 각각 저장되고, REPORT 파일은 모든 노드 프로세스가 알고리즘 수행을 종료했을 때, 호스트 프로세스에 의해 만들어지며 각 노드 프로세스에 의해 구해진 최적의 수행 결과를 저장한다.

2. 호스트 프로세스 알고리즘

시뮬레이션 시작과 함께 최초로 만들어지는 호스트 프로세스는 INPUT 파일로부터 시뮬레이션 수행에 필요한 변수들을 읽는다. 이 변수들은 시뮬레이션 수행에 필요한 기본 변수와 노드 프로세스간의 객체이주에 필요한 변수들로 구성되어 있다. 표 1은 기본 변수들에 속하는 것들을 나타낸 것이다. N_{subop}는 병렬로 수행될 서브 객체군의 수이며, 시뮬레이션에 사용될 노드 프로세스의 수와 같다. Max_Gen 은 최대 시뮬레이션 시간을 나타내는 변수이며,| P_{sub} | 는 각 노드 프로세스에 나누어질 서브 객체군의 크기이다. P_C와 P_m은 유전자 알고리즘 수행에 필요한 핵심 변수인 유전자 교배와 돌연변이 확률이고, $FREQ_{report}$는 시뮬레이션의 수행 결과를 알려주는 주기를 나타낸다. 표 2는 호스트 프로세스에 의해 만들어져서 노드 프로세스에게 전달될 변수를 보여준다. $proc_{neighbor}$ 는 해당 노드에 인접한 이웃 노드 프로세스의 인식자로서, 4개의 이웃 노드 프로세스의 인식자(ID)가 저장되어 있다. $FREQ_{migration}$와 $N_{migrants}$는 이웃한 노드 프로세스 간의 이주 시에 필요한 정보로서, 이주 주기와 이주 시에 전달되는 객체의 수를 나타낸다. 그림 7은 호스트 프로세스에 수행되는 알고리즘을 보여준다. 호스트 프로세스는 INPUT 파일로부터 시뮬레이션 변수 값을 읽어온 후, N_{subop}개의 노드 프로세스를 생성하여 노드 프로세스의 병렬 수행을 시작하도록 한다. 또한 주어진 노드 프로세스간의 연결 토폴로지(2D-torus)에 따라 각 노드 프로세스 마다 이웃 정보($proc_{neighbor}$)를 만들고, $FREQ_{migration}$과 $N_{migrants}$ 정보를 LINK 파일에 저장한다. LINK 파일에 저장된 각 노드 프로세스에 해당하는 정보는 해당 노드 프로세스에 전

표 1. ParaDAGA의 입력 변수.
Table 1. Input Parameter for ParaDAGA.

Parameters	Remarks
N_{subpop}	서브 객체군의 수
Max_Gen	최대 세대 생성 수
$\|P_{sub}\|$	서브 객체군의 크기(객체 수)
P_C	crossover rate
P_m	mutation rate
$FREQ_{report}$	수행 결과 리포트 주기
$Seed$	Random 생성기 seed

표 2. ParaDAGA의 이주 변수.
Table 2. Migration Parameter for ParaDAGA.

Parameters	Remarks
$proc_{neighbor}$	이웃 노드 프로세스 ID
$FREQ_{migration}$	이주 주기
$N_{migrants}$	이주 객체의 수

Algorithm Host()

1. Get Algorithm Parameter;
2. Forking the node processes and starting execution;
3. Make and send neighbors & migration information to each process;
4. Send the Parameters of GAs to each process;
5. Wait until all of the node processes complete the execution of Node();
6. Report the simulation results;

그림 7. 주 프로세스에서 수행되는 호스트 알고리즘.
Fig. 7. Host algorithm executed at the Main Process.

달되어 알고리즘의 수행이 시작된다. 마지막으로 호스트 프로세스는 모든 노드 프로세스의 수행이 완료될 때까지 대기(sleep) 한 후 최종 결과를 REPORT 파일에 저장한다.

3. 노드 프로세스 알고리즘

각 노드 프로세스는 호스트로부터 시뮬레이션 변수를 전달받은 직후부터 알고리즘의 수행을 시작한다(그림 8). 노드 프로세스는 $|P_{sub}|$ 크기의 초기 객체군을 독립적으로 생성한다. 생성된 초기 객체군에 대하여 Max_Gen 만큼 독립적으로 유전자 알고리즘의 기본 수행 기

Algorithm Node()

1. Receive data and Algorithm parameter from Host Process;
2. Randomly generate initial subpopulation, P_{old};
3. $P_{new} := \{\}$, $P_{migrants} := \{\}$;
4. For Gen=1 To Max_Gen
5. fitness_calculation($P_{old} \cup P_{migrants}$);
6. $P_{new} :=$ reproduction($P_{old} \cup P_{migrants}$);
7. $P_{migranys} := \{\}$;
8. crossover(P_{new});
9. mutation(P_{new});
10. If (Gen mod $FREQ_{migration}$ == 0) Then Do
11. Send copies of n migrants to the neighbor process;
12. Receive migrants(n x number of neighbor) from neighbor;
13. $P_{migrants}$ = received-migrants
14. end if
15. $P_{old} = P_{new}$;
16. If (Gen mod $FREQ_{report}$ == 0) Then Do
17. Write the fitness of the best individual to the OUT file;
18. end if
19. end for
20. Send the ending message to the host process;

그림 8. 노드 프로세스에서 수행되는 노드 알고리즘.
Fig. 8. Node algorithm executed at the Node Process.

능인, 판별 값 계산(fitness evaluation), 객체 재생성(reproduction), 유전자 교배(crossover), 유전자 돌연변이(mutation)를 수행하는데, 이 네 단계 수행 알고리즘은 DAGA에서 제안된 방식을 그대로 적용한다. 각 노드 프로세스에 의해 진화된 서브객체들은 일정 세대($FREQ_{migration}$) 진화 후, 주기적으로 인접한 노드 프로세스 간에 객체이주가 이루어진다. 이주 단계에서 각 노드 프로세스는 자신이 관리하고 있는 서브객체들 중에서 $N_{migrants}$ 만큼 선정하여 이웃한 4개의 노드 프로세스에 전달하고, 또한 이웃 노드들로부터 전달 받은 객체들을 받아들인다. 각 노드 프로세스는 자신이 관리하던 객체군과 이주한 객체를 포함한 객체군에 대해서 재생성을 수행함으로써, 새로운 서브 객체군을 형성하되, 전

체 객체군의 수는 일정하게 유지하도록 재생성을 수행한다. 이러한 과정은 다음 이주 시까지 각 프로세스 노드별로 독립적으로 진행되며, 노드별로 구해진 최적의 결과는 OUT 파일에 주기적($FREQ_{interval}$)으로 저장된다. 각 노드별로 최종 수행 단계에서는 호스트 프로세스에게 수행 종료 메시지를 보내고 수행을 완료한다.

IV. 실험 결과 및 분석

분산 객체 모델을 기반으로 구현된 ParaDAGA에서는 서브 객체 간의 객체 이주는 기존 객체군에 새로운 객체를 수용함으로써, 새로운 객체를 만들어낼 수 있는 중요한 요소가 된다. 객체이주는 이주 빈도, 이주 객체량, 이주 객체 선정 방법 등의 변수 값에 따라 구해지는 결과에 영향을 주며, 또한, 노드 프로세스에서 사용되는 서브 객체군의 수와, 객체군의 크기에 따라 구해지는 결과의 질(Quality of Solution)이 달라진다. 이러한 변수 값의 변화에 따라 ParaDAGA가 제공하는 결과의 질을 고찰하기위한 실험을 수행하였으며, 이장에서는 실험 결과와 그 결과를 분석한다.

1. 실험 환경

DAGA와 ParaDAGA의 결과의 우수성을 비교하기 위하여 표 3의 구성을 선택하였다. 선택된 구성은 순차 알고리즘인 DAGA에서 구해진 결과 중에서 다른 구성에 비해 상대적으로 결과의 우수성에 있어서 부족한 결과를 보여준 구성이다. 본 논문의 궁극적인 목적은 DAGA의 병렬 알고리즘인 ParaDAGA가 DAGA에 비해 수행시간 뿐 아니라, 구해지는 결과에 있어서도 우수성을 보장할 수 있는지를 알아보는 것이므로, 좋은 결과를 보여준 구성보다는 상대적으로 나쁜 결과를 보여준 구성에 대하여 비교해 보는 것은 타당성이 있다.

또한, 순차 알고리즘인 DAGA와의 공정한 평가를 위하여 다음의 조건을 반영하여 실험을 수행하였다.

1) DAGA 알고리즘 수행에 적용된 객체군의 크기는 ParaDAGA의 서브 객체군을 모두 합한 수 만큼 주어 두 알고리즘 수행에 사용된 전체 객체군의 크기를 같게 한다.

2) ParaDAGA의 각 노드 프로세스에서 수행되는 유전자 조작 알고리즘(유전자교환, 돌연변이, 객체 재생성)은 DAGA와 같은 알고리즘을 사용하고, 유전자 조작 변수(유전자교환 확률, 돌연변이 확률)의 값도 같은 값을 사용한다.

표 3. 실험에 사용된 구성 및 변수.
Table 3. Configurations & parameters chosen for experiments.

Configuration	Population size	Maximum Generations	T_{DATA}	$\frac{T_{DAGA}-T_{OPT}}{T_{OPT}}$ (%)
(6,4)	100	3500	1.9251	17
(6,8)	100	3500	1.3681	19
(7,4)	100	5000	2.6071	26
(7,6)	100	5000	2.2526	24

3) DAGA와 ParaDAGA를 같은 세대 수(the number of generations) 만큼 유전자 알고리즘을 수행시킨다.

모든 실험 결과 값은 각 각 독립적인 6번의 실험 결과를 평균한 값이며, 두 방식의 우수성 판별자인 T_{avg}는 네 개의 구성에서 구해진 결과 값을 평균한 값이다.

2. 이주 주기와 이주 객체 수의 영향

표 4는 이주 주기의 변화와 이주 시에 이웃 노드에 전달되는 객체의 수를 변화에 대한 실험 결과를 보여준다. 표에서 $\frac{T_{avg}-T_{opt}}{T_{opt}}$는 최적의 값에 대한 T_{avg} 값의 상대 차이를 나타낸 값이며, Δ는 DAGA에서 구해진 $\frac{T_{avg}-T_{opt}}{T_{opt}}$ 값에 대한 ParaDAGA 성능 향상을 나타낸다. 표 4에서 보는 바와 같이 ParaDAGA는 모든 구성에서 DAGA 보다 우수한 결과를 제공한다. 이주 객체의 수가 0일 경우, 즉 서브 객체간의 객체 이동이 없을 경우에도 DAGA 보다 좋은 결과를 보여준다. 이러한 실험 결과로부터 하나의 객체군을 여러 개의 서브 객체군을 만들어 독립적으로 유전자 알고리즘을 수행하면, 이들 서브 객체군들 중에 적어도 1개 이상으로부터 더 우수한 결과가 얻어질 확률이 높다는 것을 알 수 있다 또한 표 4를 보면, 이주 주기에 50 세대일 때는 이주 객체 수가 2일 때에, 그리고 이주 주기가 100일 때는 이주 객체 수가 6일 때 가장 우수한 결과를 얻을 수 있음을 보여주는데, 이로부터 서브 객체간의 이주 객체 수가 이주 주기에 따라 특정 값을 가질 때에 병렬 알고리즘의 장점을 최대한 얻을 수 있음을 알 수 있다. 이주 주기가 짧을 경우에는 이주 객체 수가 많아지면 병렬 알고리즘의 기본 원리인 분산 객체들의 독립적인 진화에 배치되어 서브 객체간의 차별성이 줄어들게 됨으로써, 하나의 객

표 4. 이주 주기와 이주 객체 수의 변화에 대한 실험 결과.
Table 4. Results with different numbers of migrants and migration interval.

(n, m)	DAGA		Migration interval								
			50 Gen.			100 Gen.			150 Gen.		
		Migrants	Migrants			Migrants			Migrants		
		0	2	6	10	2	6	10	2	6	10
(6, 4)	1.9036	1.8433	1.7831	1.8524	1.8072	1.8192	1.7951	1.8027	1.8403	1.8554	1.7831
(6, 8)	1.3207	1.3604	1.2701	1.3003	1.3102	1.3405	1.2403	1.2901	1.3102	1.2802	1.2950
(7, 4)	2.5547	2.4572	2.4606	2.5228	2.4786	2.4766	1.4489	2.4708	2.4509	2.4295	2.4747
(7, 6)	2.2092	1.9956	2.0001	1.9834	2.0201	2.0002	2.0058	1.9888	1.9883	2.0070	2.0001
T_{avg}	1.9907	1.8948	1.8784	1.9147	1.9040	1.9091	1.8725	1.8881	1.8974	1.8930	1.8882
$\frac{T_{avg}-T_{opt}}{T_{opt}}(\%)$	19.8	14.8	12.7	14.8	14.2	14.5	12.3	13.3	13.8	13.6	13.3
△(%)	-	5.0	7.1	5.0	5.6	5.3	7.5	6.5	6.0	6.2	6.5

Number of subpopulation : 4
Size of subpopulation : 100
Migrant selection strategy : Best

체에 대한 순차적인 유전자 알고리즘 적용과 같은 효과가 되기 때문이다. 반면에 이주 주기가 길 경우, 이주 객체의 수가 작은 경우보다는 큰 경우에 좋은 결과를 보여주고 있는데, 이는 각 서브 객체가 지역 최적치의 함정에서 벗어 날 수 있는 확률이 이주 객체 수가 어느 정도 큰 값을 가져야함을 나타낸다.

그림 9에서 11은 객체 이주가 각 노드 프로세스에서 진화되고 있는 객체군에 주는 영향을 보여주는 것으로서, (6,8) 구성에서 이웃 노드 프로세스로 이주되는 객체의 수가 서브 객체군 크기의 0%, 2%, 6%, 10% 일 때, 서브 객체군 내의 판별값의 분포 변화를 나타낸다. 그림에서 보는 바와 같이 이주가 없을 때에 비하여, 이주가 있는 경우에 서브 객체군을 구성하는 객체 사이의 판별 값의 최대치와 최소치의 값의 차이가 커지는 것을 알 수 있으며, 상대적으로 더 좋은 결과를 구할 수 있음을 보여준다. 그러나, 판별값 편차의 크기와 찾아낸 결과의 우수성이 비례하지는 않음을 알 수 있다. 즉, 새로운 객체의 이주는 서브 객체군의 다양성을 높여 주고, 획일적으로 수렴하는 지역 최소치에서 벗어나게 해 줌으로써, 탐색 영역을 보다 확장시키고 따라서 보다 우수한 결과를 찾아낼 확률을 높여 주는 요인임을 알 수 있다.

3. 이주 객체 선별법에 대한 영향

한 노드 프로세스 내의 객체를 이웃한 프로세스에 이주 시킬 때, 이주시킬 객체를 선정하는 방법이 ParaDAGA의 수행 결과에 영향을 주는지를 알아보기 위한 실험을 수행하였다. 본 연구의 목적은 최적의 결과를 제공하는 객체 선별법을 찾아내는 것이 아니라, ParaDAGA가 DAGA 보다 더 우수한 결과를 제공할 수 있는지를 알아 보기위한 것이므로, 일반적으로 가장 많이 적용되는 임의선정(Random) 방식과 최우수선정(Best) 방식의 두 방법에 대하여 실험을 수행하였다. 임의선정 방식은 이주할 객체를 선정할 때, 서브 객체군에서 임의로 이주 객체를 선정하는 방식이며, 최우수선정방식은 서브개체군에서 판별 값이 우수한 순서대로 객체를 선정하여 이주시키는 방식이다.

표 5는 실험 결과를 보여주는데, 최우수선정 방식이 더 우수한 결과를 보여준다. 두 방식 모두 DAGA의 결과 보다는 우수하고, 두 방식간의 결과 차이도 큼을 알 수 있다. 이 실험 결과로부터 ParaDAGA는 이주 객체 선정 방식에 매우 민감하게 반응하며, 두 방식 중에서는 서브객체내의 우수한 판별 값을 가지는 객체를 이용하였을 때 보다 우수한 결과를 제공함을 알 수 있다.

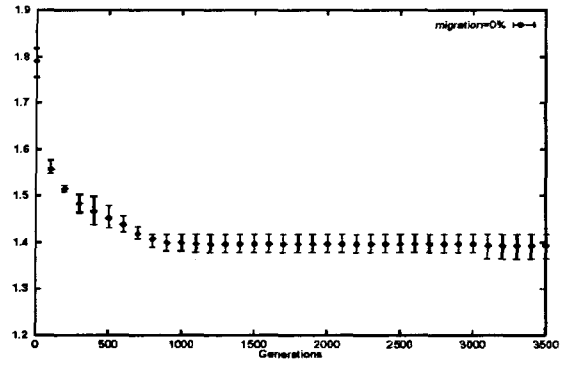


그림 9. 판별값의 최대/최소 편차: (6,8), 4 서브객체, 50 Gen. 이주주기, 0% 이주

Fig. 9. Envelope of Max/Min. Fitness value: (6,8), 4subpopulation, 50 Gen. migration interval, 0 migrants.

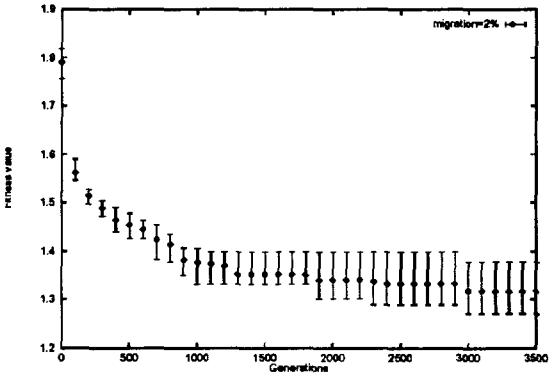


그림 10. 판별값의 최대/최소 편차: (6,8), 4 서브객체, 50 Gen. 이주주기, 2% 이주

Fig. 10. Envelope of Max/Min. Fitness value: (6,8), 4subpopulation, 50 Gen. migration interval, 2% migrants.

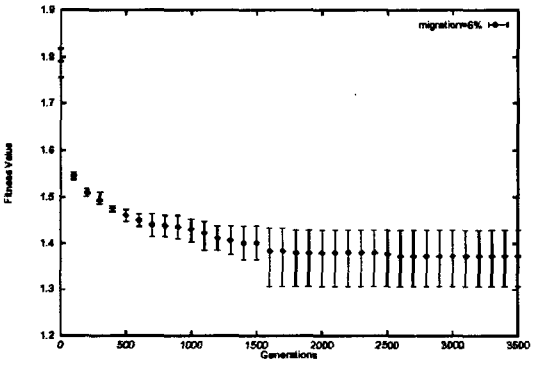


그림 11. 판별값의 최대/최소 편차: (6,8), 4 서브객체, 50 Gen. 이주주기, 6% 이주

Fig. 11. Envelope of Max/Min. Fitness value: (6,8), 4subpopulation, 50 Gen. migration interval, 6% migrants.

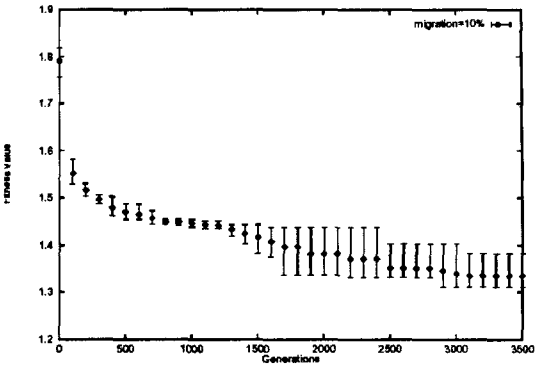


그림 12. 판별값의 최대/최소 편차: (6,8), 4 서브객체, 50 Gen. 이주주기, 10% 이주

Fig. 12. Envelope of Max/Min. Fitness value: (6,8), 4subpopulation, 50 Gen. migration interval, 10% migrants.

4. 서브 객체의 수의 영향

알고리즘 수행의 병렬도 증가에 대한 시뮬레이션 결과의 변화를 알기 위하여 서브 객체군의 수가 4일 경우와 9일 경우에 대하여 실험을 수행하였다. 병렬도의 증가란 ParaDAGA를 수행하는 노드 프로세스의 수를 증가시켰음을 의미한다.

서브 객체군의 수를 100으로 고정했을 경우에는 예상했던 대로 병렬 유전자 알고리즘은 서브 객체군의 수가 클수록, 즉 전체 객체의 수가 클수록 좋은 성능을 보여준다. 실험 결과 중의 의미 있는 것은 객체군의 수가 9일 경우 각 서브 객체군의 크기가 100일 경우와 50일 경우의 비교 결과이다. 표 6에서 보는 바와 같이, ParaDAGA는 서브 객체군의 수가 증가하면 이에 따라 성능이 반드시 증가하는 것이 아니라, 각 서브객체군의 크기가 일정 수준으로 유지되어야함을 알 수 있다. 이러한 결과는 작은 서브 객체군 크기에 의한 객체의 다양성 감소가 서브 객체수의 증가에 의한 병렬 알고리즘의 성능개선 능력에 중요한 제약을 준다는 것을 보여준다.

이 실험의 결과로부터 ParaDAGA를 병렬 컴퓨터에서 수행할 때, 각 처리기 노드에 분배될 객체군의 수를 100 이상으로 유지하지 않으면, 처리기 노드의 증가에 따른 우수한 결과의 보장을 받을 수 없다는 것을 알 수 있다.

V. 결 론

다중 디스크 시스템이 추구하는 병렬 입출력 처리 방

표 5. 이주 객체 선별법의 영향에 대한 실험 결과.
Table 5. Results with different migrants selection strategy.

(n, m)	DAGA	Migrant Selection Strategy					
		Best			Random		
		Migrants			Migrants		
		2	6	10	2	6	10
(6, 4)	1.9036	1.8192	1.7951	1.8027	1.8373	1.9743	1.9036
(6, 8)	1.3207	1.3405	1.2403	1.2901	1.3434	1.3313	1.2409
(7, 4)	2.5547	2.4766	2.4489	2.4708	2.4504	2.4033	2.5116
(7, 6)	2.2092	2.0002	2.0058	1.9888	1.9849	1.9927	1.9815
T_{avg}	1.9907	1.9091	1.8725	1.8881	1.9040	1.9254	1.9094
$\frac{T_{avg}-T_{opt}}{T_{opt}}(\%)$	19.8	14.5	12.3	13.3	14.2	15.5	14.5
△(%)	-	5.3	7.5	6.5	5.6	4.3	5.3

Number fo subpopulation : 4
Size of subpopulation : 100
Migration Interval : 100

표 6. 객체군의 수의 영향에 대한 실험 결과.
Table 6. Results with different number of subpopulation.

(n, m)	DAGA	Number of Subpopulation					
		4 (Popsize=100)		9 (Popsize=100)		9 (Popsize=50)	
		Migrants		Migrants		Migrants	
		0	6	0	6	0	6
(6, 4)	1.9036	1.8433	1.7951	1.7879	1.7590	1.3922	1.3409
(6, 8)	1.3207	1.3604	1.2403	1.3012	1.2168	1.8373	1.8132
(7, 4)	2.5547	2.4572	2.4489	2.3931	2.3576	2.5310	2.4247
(7, 6)	2.2092	1.9956	2.0058	1.9752	1.9582	2.1121	1.9694
T_{avg}	1.9907	1.8948	1.8725	1.8646	1.8229	1.9681	1.8870
$\frac{T_{avg}-T_{opt}}{T_{opt}}(\%)$	19.8	14.8	12.3	11.8	9.3	18.1	13.2
△(%)	-	5.0	7.5	8.0	10.5	1.7	6.6

Migration Interval : 100
Migrant selection strategy : Best

식은 저장될 데이터를 다중 디스크에 적절히 분산 배치하여 입출력 병렬성을 극대화해야 최대의 입출력 성능을 보장받을 수 있다. 본 논문에서는 다중 디스크 시스템에 Binary Cartesian Product File을 배치시키는 새로운 유전자 알고리즘인 DAGA를 병렬화한 ParaDA GA를 제안하였다. 버킷벡터와 디스크 사상 함수를 도입하여 다중 디스크 데이터 배치 문제를 버킷벡터 영역에서 최적의 판별 값을 가지는 하나의 버킷벡터를 찾아내는 탐색 문제로 변환하였고, 병렬 유전자 알고리즘을 탐색방법으로 제시하였다. ParaDAGA의 실험을 위하여 단일 프로세서 시스템에서 구현된 2차원 토리스 토폴로지의 상호연결망을 가지는 병렬처리 컴퓨터 시뮬레이터에 대하여 기술하였으며, 각 노드에 분산된 서브객체군에 독립적으로 유전자 알고리즘이 수행되고, 주기적으로 인접한 서브객체군 간의 객체 교환이 이루어지는 분산 객체 모델을 기반으로 하는 ParaD AGA의

세부 알고리즘을 제시하였다. 또한, 시뮬레이션 실험을 통하여 ParaDAGA의 이주 주기와 이주 객체 수의 변화에 대한 영향, 이주 객체 선별법에 대한 영향, 서브 객체의 수의 증가에 대한 영향을 분석하였다.

순차 유전자 알고리즘인 DAGA는 기존의 제안된 다중 디스크 배치 방식들 보다 우수한 결과를 제공하지만, 시뮬레이션 수행 시간이 많이 소요되어 다양한 실험을 하기에는 제약이 있었다. 반면에 ParaDAGA는 유전자 알고리즘의 병렬 수행을 통하여 시뮬레이션 시간을 줄이면서도, DAGA 보다 항상 우수하고 보다 최적에 가까운 결과를 제공하였다.

참 고 문 헌

[1] 안대영, 박규호, 임기욱, "유전자 알고리즘을 이용한 다중 디스크 데이터 배치 방식", 전자공학회논문지, 제35권 C편, 제3호, 46-58쪽, 1998.

[2] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and machine Learning, Addison-Wesley, 1989.

[3] Y. Y. Sung, "Parallel searching for binary Cartesian product files," Proc. ACM Commu nication CSC 1985 Conf., pp. 163-172, March 1985.

[4] M. Srinivas and L.M. Patnaik, "Genetic Algorithms: A survey," IEEE Computer, pp. 17-26, June 1994.

[5] H.Muhlenbein, M.Schmisch, and J.Born, "Parallel genetic algorithm as function optimizer," Proc. Int. Conf. Genetic Algorithms, pp. 271-275, July 1991.

[6] J. Lienig, "A Parallel Genetic Algorithm for Performance-Driven VLSI Routing," IEEE Trans. on Evolutionary Computation, Vol. 1, No. 1, pp. 29-49, April 1997.

[7] B. B. Prahlada Rao and R. Cc. Hansdah, "Extended distributed genetic algorithm for channel routing," Proc. IEEE Symp. Parallel and Distributed Proc., pp. 726-733, 1993.

[8] V.S. Ggordon and D. Whitley, "Serial and Parallel Genetic Algorithms as Function Optimizer," Proc. 5th Int. Conf. Genetic Algorithms, pp. 177-183, 1993.

[9] Shumeet Baluja, "Structure and Performance of Fine-Grain Parallelism in Genetic Sear ch," Proc. 5th Int. Conf. Genetic Algorithms, pp. 155-162, 1993.

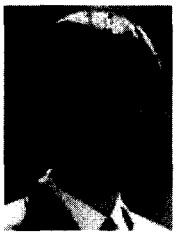
[10] K. Park, O. Frieder, and A. Sood, " A Parallel solution for the multiprocessor document allocation problem,", Proc. Int. Conf. Parallel Processing, Vol. III, pp. 119-122, August 1994.

[11] J. P. Cohoon, W. N. Martin and D.S. Richards, " A multi-population genetic algorithm for solving the K-Partition problem on hypercubes," Proc. Int Conf. Genetic Algorithms, pp.13-16, July 1991.

저 자 소 개

안 대 영(정회원)

1984년 한양대학교 전자공학과 졸업.
1986년 KAIST 전기 및 전자공학과(공학석사).
1999년 KAIST 전기 및 전자공학과(공학박사).
1986년~2002년 한국전자통신연구원(ETRI) 스토리지시스템연구팀장
현재 서원대학교 컴퓨터교육과 교수
<주관심분야 : 스토리지시스템,, 내장형시스템>



송 해 상(정회원)

1989년 전남대학교 컴퓨터공학과 졸업.
1991년 KAIST 전기 및 전자공학과 (공학석사).
2000년 KAIST 전기 및 전자공학과 (공학박사).
현재 서원대학교 컴퓨터정보통신공학부 교수
<주관심분야 : 시스템 모델링 및 시뮬레이션, 내장형 시스템, 실시간 제어>

이 상 화(정회원)

1988년 도르트문트대학(독) 컴퓨터공학과 졸업.
1994년 도르트문트대학(독) 컴퓨터공학과 (공학석사).
1998년 도르트문트대학(독) 컴퓨터공학과 (공학박사).
2000년~2002년 한국전자통신연구원(ETRI) 네트워크연구소 선임연구원.
현재 서원대학교 컴퓨터정보통신공학부 교수
<주관심분야 : 광액세스네트워크, 지능형시스템>