멀티프로세서 태스크 할당을 위한 GA과 SA의 비교

박 경 모[†]

요 약

병렬 컴퓨팅에 있어 NP-complete 문제인 태스크 할당문제에 대한 두 가지 휴리스틱 알고리즘을 제시한다. 할당문제는 분산 메모리 멀티컴퓨터의 멀티 프로세싱 노드에 다중 통신 태스크들을 최적의 매핑을 찾는것이다. 태스크들을 목표 시스템 구조의 노드들에 매핑시키는 목적은 해법 품질에 손상 없이 병렬 실행시간을 최소화하기 위함이다. 많은 휴리스틱 기법들이 만족한 매핑을 얻기 위해 채택되어 왔다. 본 논문에서 제시되는 휴리스틱 기법은 유전자 알고리즘(GA)과 시뮬레이티드 어닐링(SA) 기법에 기반을 둔다. 매핑 설정을 위한 총 계산 비용으로 목적함수를 수식화하고 휴리스틱 알고리즘들의 성능을 평가한다. 랜덤, 그리디, 유전자, 어닐링 알고리즘들을 사용하여 얻은 해법의 품질과 시간을 비교한다. 할당 알고리즘 시뮬레이션 연구를 통한 실험적 결과를 보여준다.

Comparison of Genetic Algorithms and Simulated Annealing for Multiprocessor Task Allocation

Kyeongmo Park[†]

ABSTRACT

We present two heuristic algorithms for the task allocation problem (NP-complete problem) in parallel computing. The problem is to find an optimal mapping of multiple communicating tasks of a parallel program onto the multiple processing nodes of a distributed-memory multicomputer. The purpose of mapping these tasks into the nodes of the target architecture is the minimization of the parallel execution time without sacrificing solution quality. Many heuristic approaches have been employed to obtain satisfactory mapping. Our heuristics are based on genetic algorithms and simulated annealing. We formulate an objective function as a total computational cost for a mapping configuration, and evaluate the performance of our heuristic algorithms. We compare the quality of solutions and times derived by the random, greedy, genetic, and annealing algorithms. Our experimental findings from a simulation study of the allocation algorithms are presented.

1. Introduction

We examine an important issue associated with the optimal mapping of multiple communicating tasks of a parallel program onto the multiple processing nodes of a scalable distributed-memory multicomputer. This issue leads to the mapping problem in parallel computing. The problem has been of great research interest since it was first described

※ 본 논문은 1997년 한국학술진흥재단의 공모과제 연구비의 의해 연구되었음.
† 정 회 원 : 가톨릭대학교 컴퓨터 · 전자공학부 교수
논문접수 : 1998년 7월 2일, 심사완료 : 1999년 7월 20일

by Bokhari[1]. The purpose of mapping multiple tasks into the multiprocessor nodes of the parallel architecture is the minimization of the parallel execution time without sacrificing solution quality. To minimize run time, tasks should be evenly distributed across the nodes, while the communication cost in message passing among processors should be minimized. the mapping objective is An objective function is defined to formalize the mapping goal as a minimization process. In general, obtaining an optimal solution of the mapping problem is computationally intractable; the mapping problem is known to be NP-complete[1, 6]. Therefore, heuristic approaches are commonly employed to obtain a satisfactory near-optimal solution in a reasonable time. Task allocation in distributed-memory multiprocessor systems means a mapping of a given problem to the target system. Task allocation consists of partitioning the problem into a set of disjoint subproblems (tasks) and allocating these tasks to the processors of the parallel architecture in such a way the total computational cost is minimized.

In this paper, we present two allocation algorithms. Our algorithms are based on the two types of stochastic search and optimization techniques-*Genetic Algorithms(GA's)*[7, 8] and *Simulated Annealing(SA)*[10]. These two techniques are modeled on processes found in thermodynamics, genetics, and natural evolution, and are being used in artificial intelligence systems. They have been applied successfully to difficult NP-complete problems[2, 3, 4, 8, 9, 11, 12, 13, 14, 15, 16]. This study is motivated by the fact that there is a lack of comparative studies of GAs and SA. We explore the connections between these heuristics. We also compare the mapping qualities and times derived by the genetic and annealing algorithms against those derived by a random and a greedy allocation algorithms. A simulation is developed to evaluate the performance of the allocation algorithms.

We review previous task allocation strategies. The task allocation problem and some of its derivatives are NP-complete. Therefore, heuristic algorithms that approximate optimal solutions have been developed. Some of these approaches[1, 2, 5, 9, 12, 15, 17, 18] dealt, in some manner, with the mapping of the problem graph (a set of communicating processes) onto a target architecture with a fixed interconnection topology.

Bokhari[1] proposes a mapping scheme of distributed processors that uses two input adjacency matrices to represent the problem graph (the job modules and the intercommunications) onto the system graph (the processors and the interconnections), and then applies an exhaustive pairwise exchange of two job modules. The objective function used is to maximize graph cardinality- the number of matched edges in the problem that fall on the links on the system graph. The basic assumption in the scheme is that all the problem edges are considered identical, i.e., they have the same weight. However, more general problem graph may have different weights on edges.

Lee and Aggarwal[11] extend the Bokhari's approach by incorporating a set of objective functions that accurately quantify communication overhead into the problem. The optimality of mapping the problem graph onto the system graph is evaluated by the objective functions with a more representative communication overhead measure. They has developed a mapping algorithm based on the objective functions, where they first makes an initial assignment and then iteratively apply a pairwise exchange scheme to the initial assignment. The approach is still restrictive; it utilizes a fixed path routing scheme for the network traffic.

Bollinger and Midkiff[2] formulate a two-phase mapping strategy to map a logical system onto a physical architecture using the simulated annealing algorithm, where the first phase, process annealing, assigns parallel processes onto processing nodes and the connection annealing phase schedules traffic connections on network data links so as to minimize interprocess communication conflicts. Objective functions that accurately quantify communication cost are derived to evaluate the quality of generated mapping. This effort improved upon [11] in which it utilizes the information concerning the actual routing rules.

Du and Maryanski[5] attack a variation of the mapping problem. This variation concerns the allocation of data in a dynamically reconfigurable environment. The allocation algorithm employs a set of "benefit" functions and a greedy search algorithm. The underlying execution architecture is based on a client/server model, a heterogeneous system. Although their problem closely resembles our data allocation problem, as the underlying architectural model significantly differs from our parallel execution environment, their assumptions are not relevant to our problem.

Driessche and Piessens[4] have studied the genetic algorithm for static load balancing. They present that combining genetic algorithms with simple heuristics can drastically improve the efficiency. In their paper, the usefulness of genetic algorithms in the context of dynamic load balancing is not assessed.

Mansour and Fox[12] have proposed sequential genetic algorithm for the task allocation problem in parallel computing. The cost function used is a quadratic objective function. Their problem more closely resembles our problem but our efforts are based on genetic and simulated annealing algorithms.

Hong and McMillin[9] have applied the cost measurement and error tolerance scheme based on the hill climbing nature to the composite stock cutting problem in an Intel iPSC/2 multicomputer. The asynchronous simulated annealing algorithm they proposed is based on the spatial decomposition method. Their experimental results show that the parallel algorithm results in packing densities as almost same as the sequential algorithm does with almost linear speedup.

Woodside and Monforton [18] generalize a heuristic solution based on bin-packing for finding load balanced allocations of independent tasks to multiprocessors. They introduce communicating tasks into the algorithm which are to be allocated onto a bus-connected processors and present a static allocator which could be incorporated into an automated compiler for distributed systems.

The remainder of this paper is organized as follows. Section 2 formulates the task allocation problem studied in this work. Sections 3 and 4 describe our genetic and annealing algorithm approaches to the given problem. In Section 5, we compare the solution quality derived by our heuristic algorithms against those derived by a random and a greedy allocation algorithms, and the simulation results of the algorithms are presented. Finally, the conclusions and future work are given in Section 6.

2. The Task Allocation Problem

The mapping problem we study in this work is formulated. We first define terms and notations to be used. A parallel program can be modeled by a weighted task graph, $G_t(V_t,E_t)$, in which vertices, $V_t = \{T_1, T_2,\cdots, T_t\}$, denote the tasks of a parallel program and undirected edges, $E_t = \{(T_i,T_j) \mid 1\leq i, j \leq t\}$, represent interaction between tasks. Each vertex of G_t is assigned a weight w_i which denotes the computation cost of the task T_i. Each edge is assigned a weight $w_e(T_i,T_j)$ denoting the amount of interaction between tasks T_i and T_j for $1\leq i, j \leq t$.

A parallel architecture is represented by an undirected processor graph $G_p(V_p,E_p)$ where $V_p = \{P_1, P_2,\cdots,P_p\}$ and $E_p = \{(P_m,P_n) \mid 1\leq m, n \leq p, m\neq n\}$. The vertices V_p represent the processors of the target multicomputer, and the edges E_p indicate the bidirectional communication links. In a parallel system, mapping leads to partitioning the task graph into subgraphs allocated to the processors. Tasks are partitioned into as many equally weighted clusters as the number of processors. Each cluster is then assigned by a one-to-one mapping to a processor of the multicomputer, and so the interprocessor communication cost is minimized.

Given a task graph $G_t(V_t,E_t)$ and a processor graph $G_p(V_p,E_p)$, the allocation problem consists of finding a mapping scheme $F : V_t \mapsto V_p$, which maps the t tasks of the graph G_t to the p processors of G_p, and minimizes the computation and communication cost. Let the set of vertices assigned to a cluster h be $R(h)$, i.e., $R(h) = \{T_i \in V_t : F(T_i) = h, 1 \le i \le t\}$. The computation cost (or weight w_i) of every cluster can be expressed as

$$W(h) = \sum_{T_i \in R(h)} w_i \qquad (1)$$

The communication cost of all the edges from a cluster is given by

$$C(h) = \sum_{T_i \in R(h),\, T_j \notin R(h)} w_e(T_i, T_j) \qquad (2)$$

An objective function which estimates the total parallel execution time including the computation and communication cost for a mapping configuration, is defined as

$$OF = \sum_h \Big(\sum_n W(P_n^h) + \beta \sum_n \sum_m C(P_{n,m}^h) \Big) \qquad (3)$$

$W(P_n^h)$ is the computation workload of node P_n, that is, $W(P_n^h) = Max_n(P_n^h)$ for all n, $1 \le n \le p$ and $1 \le h \le c$, where P_n^h is the number of tasks of cluster C_h allocated to node P_n. $C(P_{n,m}^h)$ is the inter-processor communication cost (matrix) between node n and node m, specified as $C(P_{n,m}^h) = Max(P_n^h M_{F(V_i),F(V_j)})$, $V_i, V_j \in C_h$ $F(V_i)$ is the processor number in the range 0 to $|V_p| - 1$, onto which the task i is mapped. β is a constant representing the relative importance of communication with respect to computation and the cost of unit computation/cost of unit communication in a machine.

3. A Genetic Algorithm Approach

Genetic algorithms simulate the survival of the fittest among individuals in nature over generations for solving a problem. Each generation consists of a population of individuals, a set of character strings. Each individual represents a point in the search space and a possible solution. We use a genetic algorithm based approach with a *distributed population* model [11, 12] which has the advantage of implicit parallelism and reduces the possibility of premature convergence. The population model involves partitioning the population into individual subpopulations. Isolated evolution with interaction among subpopulations takes place over successive generations. Our genetic mapping algorithm consists of four phases-*initialization, reproduction, crossover, and mutation*.

3.1 Representation

In the coding scheme, the set of tasks (a finite-length string) is represented by a task vector which is a sequence of integers ranging from 0 to $d - 1$. A permutation of the sequence defines an assignment of the tasks onto the nodes. A task entry D_i found at position i of a vector represents an assignment of task D_i onto node X_m, where $m = i$ *modulus* n, and n is the number of nodes. As an example, if we have a vector {1, 10, 2, 5, 8, 4, 6, 9, 7, 3, 0}, assuming that we have four processors, the vector leads to the following assignment: Node 0 contains tasks: D_1, D_8, and D_7; Node 1 contains tasks: D_{10}, D_4, and D_3; Node 2 contains tasks: D_2, D_6, and D_0; Node 3 contains tasks: D_5 and D_9.

3.2 Initialization

The first step of the algorithm is to initialize the population of individuals. In the initialization phase, a set of random permutations of the task vector is uniformly generated. Each permutation represents a possible allocation of the tasks onto the nodes.

A near-optimal allocation is generated by repetitively modifying the permutations. A permutation matrix, $P_{i,j}$ $(0 \leq i \leq n-1,\ 0 \leq j \leq d-1)$ is created. Every row of P, P_i, $(0 \leq i \leq n-1)$ is a complete permutation of all tasks D_j, $(0 \leq j \leq d-1)$. We define the mapping function, $f_i : D \mapsto X$ for any given row of P, P_i, $(0 \leq i \leq n-1)$ as $f_i(D_k) = j \bmod n$, where j is the index in row P_i of task D_k, $(0 \leq k \leq d-1)$.

〈Table 1〉 The permutation matrix

$P_{i,j}$

	0	1	2	3	4	5	6	7	8	9	10
0	1	10	2	5	8	4	6	9	7	3	0
1	8	2	4	10	6	3	5	7	0	9	1
2	4	5	8	3	0	2	9	6	1	7	10
3	5	1	2	8	7	10	6	0	9	3	4

If $n = 4$, row P_0 implies that tasks 0 through 10 are mapped to nodes 2, 0, 2, 1, 1, 3, 2, 0, 0, 3, 1, respectively.

3.3 Reproduction

The reproduction phase selects a new set of task allocations for use in the next generation using the OF. The selection process is based on the goodness/fitness value of the current permutations. The allocation with a higher value of goodness has a higher probability of producing one or more offspring in the next generation. Upon the completion of each reproduction phase, the old, poor allocations are replaced by the birth of the new, good permutations.

3.4 Crossover

The crossover phase represents the cross fertilization of permutations similar to the composition of genes from both parents in a birth. It consists of a position-wise exchange of values between each randomly paired permutations. Two-point crossover is performed on a pair of individuals by swapping contiguous segments of genes. The segment boundaries are randomly selected and are the same in both parents. Two random numbers are chosen and serve as the bounds for the position-wise exchange. Each task of the first permutation which falls within the determined bounds is swapped with the corresponding task of the second permutation, and likewise the second permutation with the first.

3.5 Mutation

The mutation phase is incorporated into the algorithm to prevent premature local convergence in the population. The mutation rate is designated by the probability of mutation. During this phase, a permutation is randomly modified with a low probability; a pair of tasks in an allocation is position-wise swapped.

3.6 Termination Condition

The termination/convergence condition is reached if all permutations are identical or if the number of generations is greater than a predetermined maximum generation limit. In our experimentation, the number of generations is obtained for five different runs and the maximum limit was set at 1500. The number of generations is implementation-dependent and must be specified carefully to obtain the best solution quality.

We now combine all the processes above to form the complete genetic algorithm for mapping.

Algorithm 1.

1. *Initialization – Randomly generate initial population of individuals.*
2. *Repeat steps 3-7 until the algorithm terminates*
3. *Evaluate goodness of individuals in population.*
4. *Reproduction – Select the string with the highest goodness value*
5. *Crossover – Pick two strings and position wise swap with a probability of crossover*
6. *Mutation – Randomly modify the string with a probability of mutation*
7. *Preserve the best solution so far.*

4. A Simulated Annealing Approach

Our simulated annealing solution to the task allocation problem is based on the SA algorithm [13] that consists of annealing steps for producing the best mapping solution. The SA algorithm used in this work that consists of annealing steps for producing the best mapping solution. The SA algorithm used in this work proceeds as follows.

Algorithm 2.

```
1. Set an initial temperature $Tem = Tem_0$;
2. Set an initial configuration $S = S_0$;
3. Calculate the cost value $C$ = calculate_C(S);
4. While <> (frozen: termination condition) do
5. Determine the vertices $V_{mv}$ to be moved;
6. While (not yet in equilibrium) do
7. Generate new configuration $S'$ = perturb(S);
8. Calculate new cost value $C'$ = calculate_C(S');
9. Calculate the cost difference $\Delta C = C' - C$;
10. if ( $\Delta C \leq 0$ )
11. then S = accept(S') update configuration;
12. else S = accept(S') with ( $e^{-\Delta C/Tem}$ ) > random(0,1);
13. End_while (with step 6);
14. Reduce temperature Tem;
15. End_while (with step 4).
```

In the algorithm, a move (perturbation) is accomplished by a random remapping of a randomly chosen configuration. A remapping that leads to a lower or identical cost is always accepted, whereas increase in the cost is only allowed with the probability $e^{-\Delta C/Tem}$ known as the *Metropolis criterion*. Acceptance probabilities of moves are controlled by a temperature Tem. The algorithm uses Eq. (3) as the cost function.

In the SA implementation, the cooling schedule policy must be specified carefully. The initial configurations are obtained by a random allocation of tasks among processors. The initial temperature is then determined such that the acceptance probability of uphill moves in the cost function is initially 0.9. Equilibrium is detected by sampling cost dynamically as the assignment is perturbed. The equilibrium at a temperature means the probability distribution of configurations has reached a steady state. Temperature is decreased by the cooling schedule in Eq. (4), making small changes in temperature. An exponential cooling schedule is used because the use of logarithmic cooling schedules requires too much computation time.

$$Tem(i+1) = C * Tem(i) \tag{4}$$

where $C < 1$ and commonly very close to 1. The constant was set to 0.98 in our implementation. $Tem(i)$ is the current temperature, that will be decreased. Eq. (4) determines the next temperature as a fraction C of the present one. SA is considered converged if one of the following two conditions is satisfied (1) if the number of accepted moves is zero, or (2) if no further progress in the mapping quality is made for a given number of annealing steps.

5. Experimental Results

We implemented and tested Algorithms 1 and 2 discussed in Sections 3 and 4 on random task graphs, respectively. The results of our mapping algorithms are favorably compared against those of a random and a greedy algorithms. The random algorithm assigns each task cluster on a randomly selected processor with no regard to computation times or communication requirements. The greedy algorithm is based on a best fit mapping strategy, where the clusters are listed in order of increasing cost, $c_0 \leq c_1 \leq \cdots \leq c_n$, and then the algorithm finds the smallest i, such that $c \leq c_i$, where c is the cost of the cluster set to be assigned. All these algorithms were implemented in C on a Sparc workstation. Our experimental findings from a simulation study of the algorithms are reported. The total value of OF of Eq. (3) is used to assess the quality of the solutions produced by the algorithms. The performance measures are the solution quality and the CPU time for algorithm execution.

Vertices V_t of task graphs are randomly generated and mapped into 16-node hypercubes. The computation and communication costs are generated randomly. Each vertex (T_i, $1 \leq i \leq t$) has an integer weight ranging from 1 to 10. T_i is connected to randomly selected vertices that will be in the range 1 to 16. Edges E_t are weighted randomly with integer values between 1 and 10. Throughout the simulations, the following GA parameter values were used: A total population size = 320, the crossover probability = 0.85, the mutation probability = 0.02, and the maximum generation limit = 1500. The parameter setting for SA was specified in the previous section.

〈Table 2〉 Comparison of four algorithms with different task graphs

$(V_t,\ E_t)$	*Random*	*Greedy*	*Genetic*	*Annealing*
(200,1000)	7677	7455	6785	6811
(300,2000)	14450	14450	12847	12939
(400,4000)	38052	36714	31754	31487

〈Table 3〉 Comparison of four algorithms with different task graphs

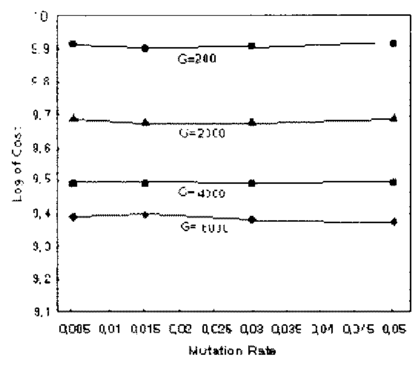
$(V_t,\ E_t)$	*Random*	*Greedy*	*Genetic*	*Annealing*
(400,1000)	9438	9055	8516	8594
(400,2000)	17808	17549	16067	16310
(400,4000)	38052	36714	31754	31487

<Table 2> compares the cost of the solutions derived by the random, greedy, genetic, and annealing algorithms for several different problem instances. The figures here are the average values of five different runs. The comparison results in <Table 2> represents the best solutions obtained by the four algorithms in these multiple runs. <Table 3> also compares the cost of the solutions derived by the four algorithms for different problem instances. However, the same value (400) of V_t was used for each problem with different E_t.

〈Table 4〉 Time (in sec.) of four algorithms with different task graphs

$(V_t,\ E_t)$	*Random*	*Greedy*	*Genetic*	*Annealing*
(400,1000)	42	96	144	187
(400,2000)	74	161	268	314
(400,4000)	82	180	453	565

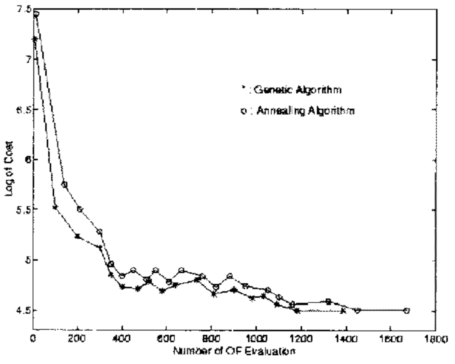
<Table 4> shows the execution time taken. The presented results show that the random and greedy algorithms yield solutions of lower quality than those of the genetic and annealing heuristics, but they are faster at the expense of solution quality. That is, faster techniques produced lower quality solutions. For instance, in the (400, 4000) problem, the genetic approach gave the best result, improving by about 19% over the random and 15% over the greedy. The results indicate that the genetic algorithm enjoys the property of using the population and its implicit parallelism, and thus this will enable the GA approach to get better performance. It should be noted that in this experiment, the genetic and annealing algorithms were not optimized for time, and aimed for the mapping solution quality. In optimization problems, the performance measure of the solution quality is of greater importance.



(Fig. 1)

(Fig 1) demonstrates the relatively negligible effects of mutation on the allocation derived for (400, 4000). The cost for different generations. The values of mutation rates used were in the range 0.005-0.05. The results indicate that lower cost of the GA is achieved with the larger number of generations and the impact of varying the mutation rate is minimal. For additional experimental results, see [14].

The performance obtained using GA and SA approaches is comparable. From Table 2, 3, and 4, we can see that the solution quality by the genetic method are slightly better in smaller problems but the annealing solution is better in a larger problem. Genetic algorithms search from a population of points, while only a single point in simulated annealing is perturbed. The time consuming evaluation of the objection function can be done in parallel for a whole population, as can the reproduction of offspring individual solutions. The implicit parallelism of genetic search tends to evolve good solutions in a shorter time. The genetic search time can be reduced by parallelizing the sequential GA on multiprocessor systems. It has been shown that, in prior studies[11] near linear speedups can be observed by an asynchronous parallel GA. In distributed-memory multicomputers, the scalability of SA is lower than that of GA due to inherent global synchronization involved.



(Fig. 2)

(Figure 2) illustrates the behavior of the genetic and annealing algorithms. The cost of the solutions is plotted against the number of OF evaluations. Each of the algorithms approches a suboptimal solution when the fluctuating cost is stabilized. The genetic approach gives greater initial improvements than the annealing one. The annealing runs tend to converge late in comparison with the genetic. The results show that the genetic approach makes initial improvements larger and the progress of annealing is consistent with the larger number of evaluations.

6. Conclusions

We have studied the task allocation problem (an NP-complete problem) in parallel computing. We have presented two heuristic solutions based on genetic algorithms and simulated annealing. We evaluated the performance of our allocation algorithms by orders of magnitude, and compared the quality of solutions derived by the random, greedy, genetic, and annealing algorithms. Experimental results from a simulation study of the algorithms were obtained. The solution quality derived by the genetic or annealing algorithm was found to be superior to those of either the random or the greedy. The performance obtained using the genetic or annealing approaches is comparable. The genetic approach makes initial improvements greater and the annealing process is consistent with the larger number of iterations. In our ongoing research, we are developing a hybrid algorithm which combines the benefit of genetic and annealing algorithms. Related research works along with this direction can be found in [15, 16].

References

[1] S. H. Bokhari, "On the Mapping Problem," IEEE Trans. on Computers, Vol.C-30, No.3, pp.207-214, 1981.

[2] S. W. Bollinger and S. F. Midkiff, "Processor and Link Assignment in Multicomputers Using Simulated Annealing," Proc. Int. Conf. Parallel Processing, pp.1-7, Aug. 1988.

[3] 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(ICGA), pp.244-249, Jul. 1991.

[4] R. V. Driessche and R. Piessens, "Load Balancing with Genetic Algorithms," Proc. Conf. Parallel Problem Solving from Nature, North-Holland, Amsterdam, pp.341-350, 1996.

[5] X. Du and F.J. Maryanski, "Data Allocation in a Dynamically Reconfigurable Environment," Proc. Int. Conf. Data Engineering, pp.74-81, 1998.

[6] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-

completeness, W. H. Freeman and Co. San Francisco, CA., 1979.

[7] D. E. Goldberg, Genetic algorithms in Search, Optimization and Machine Learning, Addison Wesley, Reading, MA., 1989.

[8] J. H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, (Also, MIT Press), 1975 & 1992.

[9] C.-E. Hong and B. M. McMillin, "Relaxing Synchronization in Distributed Simulated Annealing," IEEE Trans. Parallel & Distrib. Systems, Vol.6, No.2, pp.189-195, 1995.

[10] S. Kirkpatrick, C. D. Gelatt, Jr. and M. P. Vecchi, "Optimization by Simulated Annealing," Science, Vol.220, pp.671-680, May, 1983.

[11] S.-Y. Lee and J. K. Aggarwal, "A Mapping Strategy for Parallel Processing," IEEE Trans. Computers, Vol.C 36, No.4, pp.433-442, 1987.

[12] N. Mansour and G.C. Fox, "A Hybrid Genetic Algorithm for Task Allocation in Multicomputers," Proc. ICGA, pp.466-473, Jul. 1991.

[13] K. Park, O. Frieder, and A. Sood, "A Parallel Solution for the Multiprocessor Document Allocation Problem," Proc. Int. Conf. Parallel Processing, Vol.3, pp.119-122, 1994.

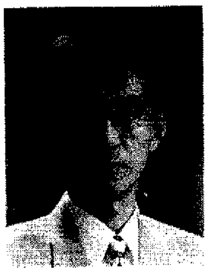
[14] K. Park, O. Frieder and A. Sood, "Data Allocation in Distributed Memory Systems: A Genetic Approach," Proc. Parallel & Distributed Processing Techniques and Applications, Vol.3, pp.1190-1201, 1996.

[15] K. Park, "A Comparative Study of Mapping Heuristics," ACM SCS, High Performance Computing, Vol.1, pp.278-283, Apr. 1996.

[16] K. Park and C.-E. Hong, "Performance of Heuristic Task Allocation Algorithms," Journal Natural Science, CUK, Vol.18, pp.145-155, 1997.

[17] J. Xu and K. Hwang, "Mapping Rule-Based Systems onto Multicomputers Using Simulated Annealing," J. Parallel & Distributed Computing, Academic Press, Vol.13, pp.442-455, 1991.

[18] C. M. Woodside and G. C. Monforton, "Fast Allocation of Processes in Distributed and Parallel Systems," IEEE Trans. PADS, Vol.4, No.2, pp.164-174, 1993.



박 경 모

e-mail : kpark@eos.cuk.ac.kr

1980년 중앙대학교 전자계산학/컴퓨터공학과 졸업(학사)

1983년 서울대학교 계산통계학과 전산과학전공(이학석사)

1990년 美 New Jersey Institute of Technology, 컴퓨터정보학과(컴퓨터과학석사)

1994년 美 George Mason University 정보공학부(컴퓨터공학박사)

1996년~현재 가톨릭대학교 컴퓨터공학과 조교수

관심분야 : 분산/병렬처리, 컴퓨터시스템 성능평가, 휴리스틱 및 최적화 기법 등