클러스터 문서할당을 위한 휴리스틱 기법에 관한 연구*

박 경 모

가톨릭대학교 컴퓨터공학부

A Study on Heuristic Methods for Clustered Document Allocation

Kyeongmo Park

Computer Science and Engineering School, The Catholic University of Korea

요약

본 논문에서는 병렬 정보검색 시스템에 있어 클러스터 문서할당을 위한 두 가지 휴리스틱 기법을 제시한다. 효율적 문서할당에 관한 매핑 문제를 정의하고 유전알고리즘과 모의냉각기법에 기반을 두는 휴리스틱 매핑 알고리즘을 기술한다. 알고리즘 성능실험과 관련하여 시뮬레이션을 통한 다른 할당 알고리즘과 비교평가한 결과 개선된 성능을 얻을 수 있었다.

1 Introduction

As the volume of data accessible online is increasing, conventional information retrieval (IR) systems will not respond to users' queries within an acceptable time period. To meet the demands needed to deliver acceptable response times in the wake of large IR databases, parallel IR systems are increasingly being used IR systems based on parallel multiprocessor architectures can use vast computational resources more efficiently by spreading work across the multiple processing nodes However, a multiprocessor architecture introduces the problem addressing the optimal mapping of a clustered collection of documents onto the multiple nodes, namely the Multiprocessor Document Allocation Problem [1] A poor document mapping onto the multiprocessor results in high access and retrieval times. In general obtaining an optimal solution of the mapping problem is computationally intractable: the problem is known to be NP-complete Therefore, heuristic approaches are commonly employed to obtain a satisfactory near-optimal solution in a reasonable time.

In this paper, we present two heuristic allocation methods based on the two types of stochastic search and optimization approaches. Genetic Algorithms (GA's) and Simulated Annealing (SA). 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 [2]. We also compare the mapping qualities 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

The remainder of this paper is organized as follows. Section 2 formulate the document allocation problem studied in this work Sections 3 and 4 describe our genetic and annealing algorithm techniques to the test 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 sample simulation results of the algorithms are presented Finally, the conclusions and future work are given in Section 6

2 Problem Formulation

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, $E_t = \{(T_i, T_j) | 1 \leq i, j \leq t\}$, 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 $w_c(T_i, T_j)$ denoting the amount of interaction between tasks T_i and T_j for $1 \leq i, j \leq t$ A

* 이 논문은 1997년 한국학술진흥재단 공모과제 연구비에 의하여 연구되었음

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 $E_p = \{(P_m, P_n) | 1 \leq m, n \leq p, m \neq n\}$ The vertices V_p represent the processors of the target multicomputer, and the edges E_p indicate the bidirectional communication links.

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 . $V_t \mapsto V_p$, 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., $R(h) = \{T_i \in V_t : F(T_i) = h, 1 \leq i \leq t\}$. 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

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

$W(P_n^h)$ is the computation workload of node P_n, that is, $W(P_n^h) = Max_n(P_n^h)$ for all n, $1 \leq n \leq p$ and $1 \leq h \leq c$, 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.

3 A Genetic Algorithm

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

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_n, where $n = i$ *modulus* p, and p is the number of processors.

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.

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.

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

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 The termination 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 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 to 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 Simulated Annealing

Our simulated annealing solution to the mapping problem is based on the SA algorithm [3] 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_{mvs} *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

$$Tem(i+1) = C * Tem(i) \quad (4)$$

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. 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

(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 1: Comparative Evalution of mapping algorithms

(V_t, E_t)	*Random*	*Greedy*	*Genetic*	*Annealing*
(200, 1000)	42	96	144	187
(300, 2000)	74	161	268	314
(400, 4000)	82	180	453	565

Table 2: Time (in sec.) of four mapping algorithms

참고문헌

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

[2] K Park, "A comparative study of mapping heuristics,' *ACM Simulation Conf High Performance Computing*, Vol. I, pp. 278-283, Apr 1996.

[3] K. Park and C.E Hong, ' Performance of heuristic algorithms," *Journal of Natural Science*, CUK, **18** pp 145-155, Dec 1997.