이종의 다중컴퓨터에서 태스크 할당을 위한 효율적인 알고리즘

서 경 룡[†] · 여 정 모[††]

요 약

본 논문은 서로 다른 성능을 가진 프로세서들로 구성된 다중컴퓨터 시스템에서 태스크의 할당에 관한 문제를 다룬다. 다중컴퓨터 시스템의 성능을 최대로 발휘하기 위해서는 분산구조를 가진 프로그램 모듈들을 실행시간을 최소화하도록 각 프로세서에 할당하여야 한다. 이러한 할당방법을 태스크의 균등할당이라 하는데 부하가 적절하지 못한 프로세서는 제 성능을 발휘하지 못하고 전체 시스템의 성능을 저하시키기 때문에 태스크를 균등하게 할당하는 것이 성능향상을 위한 좋은 방법이다.

이러한 태스크 할당문제를 해결하기 위하여 본 논문에서는 비 균등 할당의 비용을 수식화 할 수 있는 새로운 목적함수를 제시하였다. 제안된 목적함수를 사용하여 태스크 할당문제를 통신비용과 작업비용, 그리고 비 균등 할당비용의 합을 최소화하는 문제로 단순화 시켰다. 이렇게 변화된 문제는 NP-hard의 문제이므로 최적에 근사한 할당을 구하는 $O(n^2m)$의 복잡도를 가지는 휴리스틱 알고리즘을 제안하였다. 이때 m, n은 각각 태스크와 프로세서의 개수이다.

An Efficient Task Assignment Algorithm for Heterogeneous Multi-Computers

Kyung-ryong Seo[†] · Jeong-mo Yeo[††]

ABSTRACT

In this paper, we are considering a heterogeneous processor system in which each processor may have different performance and reliability characteristics. In order to fully utilize this diversity of processing power it is advantageous to assign the program modules of a distributed program to the processors in such a way that the execution time of the entire program is minimized. This assignment of tasks to processors to maximize performance is commonly called load balancing, since the overloaded processors can perform their own processing with the performance degradation.

For the task assignment problem, we propose a new objective function which formulates this intervening cost. Thus the task assignment problem is to be carried out so that each module is assigned to a processor whose capabilities are most appropriate for the module, and the total cost is minimized that sum of inter-processor communication cost and execution cost and imbalance cost of the assignment. To find optimal assignment is known to be NP-hard, and thus we proposed an efficient heuristic algorithm with time complexity $O(n^2m)$ in case of m task modules and n processors.

1. Introduction

A heterogeneous multi-computer system is considered to be a set of programmable processors, each with private memory, interconnected to some extent by communication links [6]. The heterogeneity in a computing system is not an entirely new concept. Several types of special-machines have been used to provide

† 종신회원 : 국립부경대학교 컴퓨터공학과 조교수
†† 정 회 원 : 국립부경대학교 전자계산학과 부교수

논문접수 : 1997년 11월 4일, 심사완료 : 1998년 3월 14일

specific services for improving system throughput.

A task to be run on a distributed system consists of m modules. Each of the modules comprising a task will execute on the one of n processors and communicate with some oth -er modules of the task. Furthermore, a dist -ributed program is defined as a program that consists of several program modules or tasks that are free to reside on any processor in the system.

Because of high penalties for communication, the practical solution for using a heterogeneous system is to do as little communication between processors as possible, relegating such commun -ication to file transfers before and after major blocks of computation are run. This category of scheduling problems have been traditionally formulated as a task assignment problem[1, 3,4].

A task allocation problem is a difficult problem even without timing constraints. For example, finding optimal assignment of tasks with an arbitrary communication graph to four or more processors with difference speeds is known to be NP-hard. Considerable efforts have been spent on more restrictive allocation problems or on developing heuristic algorithms to find suboptimal solutions. Most of the results can be extended and applied to find suboptimal solutions.

Stone developed network flow algorithms[14] to allocate tasks with arbitrary communication patterns in dual-processor and three-processor systems. Lee and et al. extended Stones result to a linear-array structured system[7]. Recently, the task assignment problem on host satellite systems and tree structure systems using simil ar approach are developed by Seo and et al[8, 9].

Bokhari developed a dynamic programming algorithm to allocate tasks that form a tree to a system with an arbitrary number of processors. The time complexity of his algorithm is $O(nm^3)$ [5]. His approach was extended by Towsley[15] to handle tasks with series-parallel precedence graph.

The above approaches consider heterogen-eous processors. However, these approaches attempt to minimize the total execution cost and the communication cost of tasks. They do not attempt to balance the load of the processors[2].

Efe, et al. developed a heuristic allocation algorithm[10,11] to balance processor load and to minimize communication cost. His algorithm consist of two phases. First, tasks are clustered with each other to optimize the communication cost and each cluster of tasks is assigned to a processor. Then, tasks are shifted from overloaded to underloaded processors in order to meet load balance constraints.

The algorithm is repeated until a satisfact -ory degree of load-balancing is archived. This algorithm, however, fail to distribute the tasks evenly when the processors in the system are not similar.

Lo developed heuristic algorithms[12] which incorporate a cost function to maximize the concurrent approaches that can also be applied to hard real-time systems in the same ways as described above.

In this paper, we present an efficient heuristic task allocation algorithm. The

allocation is done before actual run time of the application problem. It is assumed that the execution cost of the subtasks and the IPC cost, which arises due to the interacting modules residing on different processors. It distributes the modules of an application problem as evenly as possible among the processors and tries to minimize the IPC cost. The method can be applied to partitioned programs of any number of modules and to multi-computers with any number of processors and any point-to-point interconnection network.

2. Task Allocation Problem

2.1 Problem model

A multi-computer system can be represented by an undirected graph called a processor graph, $G_P = (V_P, E_P)$, where V_P is the set of processors in the system and $E_P \subseteq V_P \times V_P$ is the set of edges representing communication links between processors.

Similarly, each task can be described by an undirected graph called the task graph, $G_T = (V_T, E_T)$, where V_T is the set of nodes representing a task of the job, and $E_T \subseteq V_T \times V_T$ is the set of edges representing the intertask communication between the two task nodes connected by the edge. When there is an edge between two task nodes in G_T, the two tasks are said to be related to each other.

In our case, the interconnection of processors is modeled by a cost matrix C with a typical element c_{ij} denoting the volume of communication between tasks i and j. If the link $(i,j) \notin E_T$ then $c_{ij}=0$. Also we take $c_{ii}=0$, for all i.

The communication cost can be thought of as being composed of two parts: a static part that takes into account the total number of bytes transferred in a single execution of tasks i and j, and a dynamic part that accounts for the frequency of process execution and communications related queueing delays.

Note that whereas the static communication cost can be precisely accounted for by inspection of the tasks cost, the dynamic part has to be estimated based on gathered statistical information and is dependent upon the allocation itself. Next we model the interconnection of processors through a delay matrix D where the typical element d_{kl} denotes the communication delay for sending a byte from a processor k to a processor l.

If processors k and l are neighbors in the processor graph, then d_{kl} reflects the cost of point to point or multiple access communication.

If the processors are not neighbors, but there is a path form k to l in the processor graph, then d_{kl} reflects the cost of intraprocessor message sending per unit message length. If a particular allocation assigned task i on processor k and task j on processor l, then the communication cost for this particular assignment is taken to be $c_{ij} \cdot d_{kl}$.

Each task i represents a load l_{ik} on processor k which reflects CPU time this task demands.

For given processors and attached I/O devices, we can calculate the load of a specific process, if we let the process run on the processor and measure the CPU execution time e. In addition, we have to measure the rate at which this process is asked to execute.

The problem involves the development of

task allocation models for the heterogeneous computing system. The model must allocate tasks among the processors to archive the following goals:

- Allow specification of a large number of constraints to facilitate a variety of engineering application requirements.
- Balance the utilization of individual processors.
- Minimize the interprocessor communication cost.

The design of a mathematical model for task allocation for a heterogeneous computing system involves the following steps:

- Formulate the cost function to measure the interprocessor communication(IPC) cost and the processing cost.
- Formulate a set of constraints to meet the diverse requirements.
- Derive an iterative algorithm to obtain a minimum total cost solution.

2.2 Cost function

The cost function is formulated as the sum of the IPC cost and the processing cost.
IPC cost is a function of both task coupling factors and interprocessor distance. Coupling factor c_{ij} is the number of data units transferred from task i to task j. Interprocessor distance d_{kl} is certain distance related communication costs associated with one unit of data transferred from processor k to processor l. If tasks i and j are assigned to processors k and l, respectively, the interprocessor cost is $c_{ij} \cdot d_{kl}$. If $k = l$ then $d_{kk}=0$. The unit of IPC cost is application dependent. For example, the unit of c_{ij} is word and d_{kl} is \$/word, the IPC unit is dollars.

Processing cost q_{ik} represents the cost to process task i on processor k. It can be used to control the processor assignment. For example, if task i must not be executed on the processor k, a very large value can be assigned to q_{ik} to inhibit the assignment.
The assignment variable is defined as follows:

$$X_{ik} = \begin{cases} 1, & \text{if task } i \text{ is assigned to processor } k. \\ 0, & \text{otherwise.} \end{cases}$$

The total cost for processing the tasks is stated as

$$\sum_i \sum_j (wq_{ik} X_{ik} + \sum_i \sum_j c_{ij} d_{kl} X_{ik} X_{jl}).$$

The normalize constant w is used to scale processing cost and IPC cost to account for any difference in measuring units.

In general, we are considering a heterogeneous processor system in which each processor may have different performance and reliability characteristics. In order to fully utilize this diversity of processing power it is advantageous to assign the program modules of a distributed program to the processors in such a way that the execution time of the entire program is minimized.

This assignment of tasks to processors to maximize performance is commonly called load balancing. Each processor k sets an upper bound U_k on the total load that can be allocated, beyond which this processor enters the nonlinear part of its load versus throughput curve and becones saturated.

Total load on a processor is the sum of execution costs of the modules assigned to it. Then the load balancing constraints can be written as:

$$\sum_i l_{ik} X_{ik} \leq U_k \text{ for all } k$$

The above load constraint is hard to treat since it's nonlinear property. The overloaded processors can perform their own processing with the performance degradation. If a processor not run on the sutable load, the amount of lower or upper it's load capability is a measure of loss or price paid because of imbalance, compared to the case when each processor has the suitable load.

For a given assignment the load imbalance cost is formulated as follows:

$$COST_{load} = \sum_{k=0}^{m} \left| \sum_{i=0}^{n} l_{ik} X_{ik} - U_k \right|.$$

Thus the processor allocation of program modules is to be carried out so that each module is assigned to a processor whose capabilities are most appropriate for the module, and total cost is minimized that is sum of IPC cost and execution cost and imbalance cost of the assignment, i.e.,

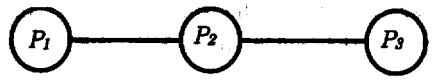
$$COST_{total} = COST_{exec} + COST_{IPC} + COST_{load}$$

$$= \sum_{i}\sum_{k}(wq_{ik}X_{ik}) + \sum_{i}\sum_{j}\sum_{k}\sum_{l}(c_{ij}d_{kl}X_{ik}X_{jl}) + \sum_{k}\left|\sum_{i}l_{ik}X_{ik} - U_k\right|$$

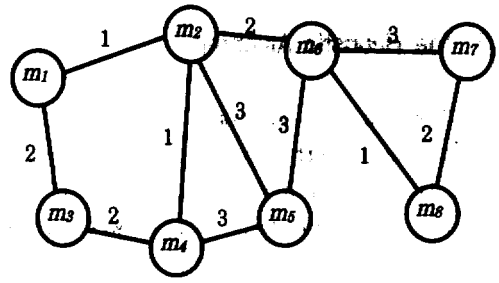
2.3 Task allocation Example

The Fig. 1 and shows an example program graph in a distributed system. The nodes in the program graph represent modules, and the links represent intermodule communication patterns. The numbers on the branches, called the branch weights, represent the cost of communication between modules if the modules are not resident modules are assumed to be zero when the pair of modules can coexist. The costs are normally given in units of time or dollars, and the units must be the same units to express execution costs.

The Table 1 shows the execution costs of the program modules. An infinite cost indicates



a) processor graph



b) Task graph

(Fig. 1) Processor graph and Task Graph

〈Table 1〉 Execution cost table

	P_1	P_2	P_3	
m_1	5	∞	∞	attached to P_1
m_2	4	3	3	
m_3	2	2	4	
m_4	4	4	5	
m_5	2	3	∞	
m_6	3	3	2	
m_7	∞	4	4	
m_8	∞	∞	1	attached to P_3

(Fig. 2) An allocation example with total cost 43.

that a module cannot be executed on that processor. We have shown in this example a situation in which some modules run faster on processor 1 and some modules run faster on processor 2 and others run faster on processor 3. For any given module assignment, the cost

of the assignment is the sum of the execution costs from the table plus the sum of the intermodule communication costs for those modules that are not coexist.

3. The Heuristic Algorithm

3.1 Definitions and assumptions

In almost all other heuristic approaches, a search is made for a pair of modules with the maximum communication cost between them. Such a module pair is assigned to a processor with the intention of minimizing the IPC cost. But, in several cases, assigning a pair of modules with the largest communication cost to a processor would not result in a reduction of IPC cost. This fact is evident when a module is interacting with many other modules.

In our algorithm, therefore we do not search for such pairs of modules, but for single modules called maximally linked modules, as defined below:

Definition:

The sum of intermodule communication cost of m_k with other modules is called the link capacity of the module and a module m_k in a program graph is called a maximally linked module if the link capacity is larger than the link capacity of an other module, i.e.,

$$\sum_i c_{ki} > \sum_h c_{hk} \text{ for all } h \neq k \text{ such that } c_{hk} \neq 0$$

where c_{ij} is the IPC cost between the module m_i and m_j when m_i and m_j are assigned to different processors. Thus, a maximally linked module can be easily determined by finding the sum of the labels of all edges incident on a node. Since a maximally linked module has maximum communication with the interacting modules, it would be advantageous to form clusters around such modules. Thus, a maximally linked module will tend to absorb its neighboring modules. The maximally linked modules represent maxima of intermodule communications in the given program graph. Some modules may have to be assigned to specific processors, to exploit their unique capabilities.

In this respect modules are divided into three categories, as follows:

(a) attached modules that can only be assigned to certain processors

(b) modules that can be assigned to any one of a certain set of processors;

these are also considered as attached modules

(c) modules that can be assigned to any processor.

3.2 Heuristic algorithm

Our algorithm starts by assigning one or more modules to each processor. These could be the attached modules of the processors. If a processor dose not have any attached module, then we assign one of the maximally linked module to it. Having assigned an initial module to each processor, other modules which have maximum data exchfange with the already assigned modules on a processor are chosen for allocation.

Thus, the module clusters are formed around the attached modules or maximally linked modules.

The number of clusters is equal to the number of processors. This process is to continued until the total cost becomes minimum. The minimally linked modules are used to adjustment load condition. The

algorithm is as follows:

The algorithm
Step 1: Order all modules of $M + (m_1, \ldots, m_n)$ in decreasing order of link capacity.
Step 2: Assign initial module to each processor.
Processors having attached modules are assigned these modules.
These modules are removed in the list M.
Step 3: For a processor P_i, calculate unbalance cost.
Step 4: **While** (List M is not empty) do
Get a module m_k from M and allocate to the processor P_i which can result the minimum cost.
Update the unbalance cost of P_i.
End(While)
Step 5: Order all modules of $M = (m_1, \ldots, m_n)$
Step 6: Repeat
Choose a module m_k which is assigned to processor P_i from M and a processor P_j such that the cost improvement is maximum.
If such a m_l exists, move it to P_k.
Until no m_l was chosen
(★End of algorithm★)

Step 1 and Step 5 list modules in preferred orders. Step 3 calculates the unbalance cost of the processors. Step 4 tries to assign the module at the processor is the most cost efficiently. Step 6 tries to eliminate as much cost of an assignment as possible. Using an efficient sorting algorithm, Step 1 and Step 5 can be done in $O(n \log n)$ in the worst case. Step 4 needs $O(n^2m)$ comparisons and Step 6 needs $O(nm^2)$ comparisons. Since n is typically greater than m, the time complexity of the algorithm is $O(n^2m)$.

3.3 An illustrative example

In this example, the distributed computer system consists of three processors with the topology shown in Fig. 1. Table 1 gives the load capacity of the processors and attached modules. Module allocation is started by initially assigning the attached modules to their respective processors:
module 1 to p1 and module 7 to P3 and ordered set M = (m6, m2, m4, m3, m7) is obtained. In order to allocate a module to processor properly, A good estimation cost function is required.

If m_i is selected to allocate to P_k, we calculate an estimated allocation cost $COST_{ik}$. An estimated cost $COST_{ik}$ consist of the three cost $EXEC_{ik}$, IPC_{ik} and $LOAD_{ik}$.
Each cost is calculated as follows.

$$EXEC_{ik} = l_{ik}$$

$$IPC_{ik} = \sum_{i}^{m}\sum_{j}^{n} c_{ij}d_{kl}X_{ik}X_{jl}$$

$$LOAD_{ik} = \sum_{i}^{n} |l_{ik}X_{ik} + U_k| - \sum_{i}^{n} |l_{ik}X_{ik} + l_{ik} - U_k|$$

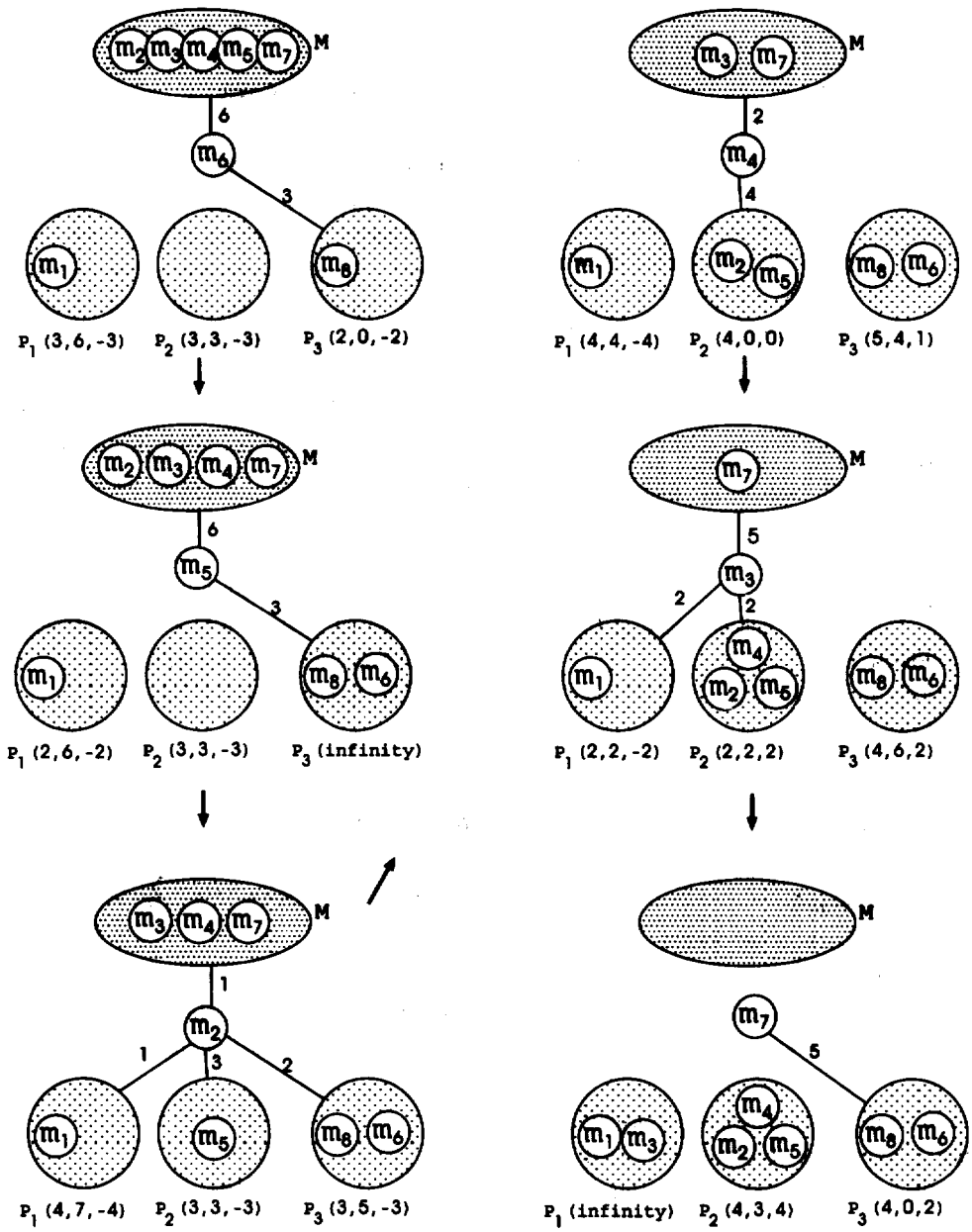
In this case, m_6 is maximally linked module, thus m_6 is selected for the next allocation and the estimated cost is calculated as follows:

$$EXEC_{61} = 3,$$
$$IPC_{61} = 6,$$
$$LOAD_{61} = -3,$$

$$EXEC_{62} = 3,$$
$$IPC_{62} = 3,$$
$$LOAD_{62} = -3,$$

$$EXEC_{63} = 2,$$
$$IPC_{63} = 0,$$
$$LOAD_{63} = -2,$$

Since the sum of costs of $COST_{63}$ is



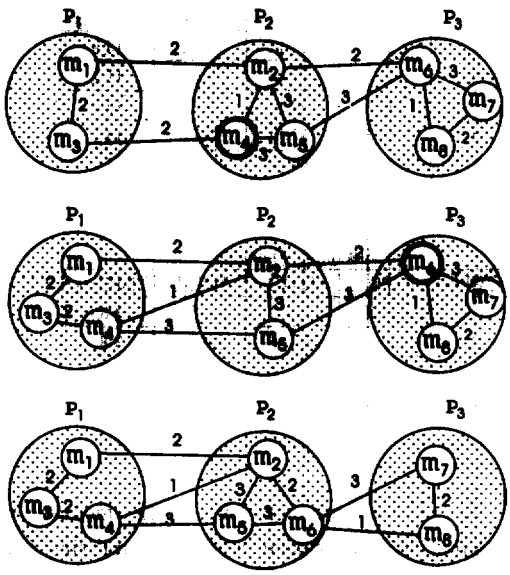
(Fig. 3) Snapshots of stepwise execution of the algorithm

minimum, thus m_6 is allocated to P_3.

This process is continued until list M is empty. All these steps involved in module allocation are illustrated in Fig. 3.

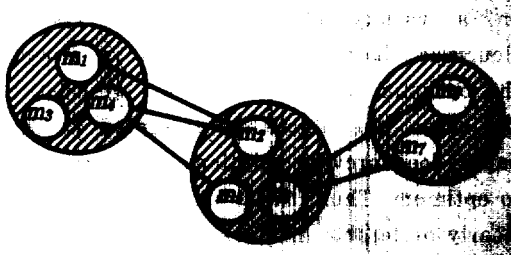
If all modules are allocated to the processors, the reallocation process is started(Step 5, Step 6). First, the minimally linked module m1 is selected, but it is attached module. Thus the next module m7 is selected and reallocation cost is calculated as

similar with previous described manner.



(Fig. 4) Reallocation process

All these steps involved in module reallocation are illustrated in Figure 4. In this case, the module m_4 is reallocated to P_1 from P_3 and the module m_6 is reallocated to P_2 from P_3. The final clusters allocated to each processors are shown in Figure 5.



EXEC = 11 + 8 + 5 = 24
IPC = 6 + 4 = 10
LOAD = 1 + 0 + 0 = 1
TOTAL = 35

(Fig. 5) Final allocation with cost 35

4. Simulation results and Conclusions

4.1 Simulation results

To evaluate the effectiveness of the proposed algorithm, simulations were run on a Sun Sparc Station20 running Solaris 2.4.

The results in Table 2 show the general diff -erences between the Sarje's algorithm[13] and our proposed algorithm. We performed 600 sim -ulations for fully connected computer system containing 3-7 processors. A large number of program graphs were considered for evaluating the performance of our algorithm. The execution cost of subtasks were picked randomly in the range from 5 to 50, and IPC costs were chosen in the range from 0 to 20.

The algorithm in [13] can get a good, but not an optimal allocation for a homogeneous and fully connected multi-computer system. Thus simulations are evaluated for the homogeneous fully-connected multi-computer systems.

The results in Table 3 show that the algorithms are pratical even for large, heterogeneous, arbitrary structure computer systems. We perf -ormed 400 simulations for a hypercube struct -ured computer system containing 8-128 process -ors. A large number of program graphs were considered for evaluating the performance of our algorithm. The execution cost of modules were picked randomly in the range from 2 to 100, and IPC costs were chosen in the range from

A simulated annealing approach is based on an analogy between annealing process in which a meterail is melted and cooled very slowly and the solution of a difficult combinatorial optimization problem. The running times

〈Table 2〉 Simulation results1

m	Our Algorithm					Sarje's Algorithm				
	EXEC	IPC	LOAD	TOTAL	TIME(ms)	EXEC	IPC	LOAD	TOTAL	TIME(ms)
3	481.2	121.3	62.2	664.7	13.0	487.3	120.6	74.2	682.1	10.9
4	485.5	134.7	65.6	695.8	14.3	492.2	135.3	77.4	704.9	11.2
5	483.6	146.9	68.5	699.0	15.8	491.5	148.1	80.2	719.8	12.4
6	480.1	161.4	76.7	718.2	15.1	484.7	163.5	87.6	735.8	14.7
7	481.9	183.3	80.9	746.1	16.3	486.7	185.3	90.8	762.8	17.5

〈Table 3〉 Simulation results2

m	Our Algorithm					Simulated Annealing				
	EXEC	IPC	LOAD	TOTAL	TIME(ms)	EXEC	IPC	LOAD	TOTAL	TIME(ms)
8	863.5	186.3	104.2	1154.0	26.2	847.6	185.1	115.2	1147.9	3.85
16	860.7	321.4	118.4	1300.6	34.1	843.4	325.4	130.4	1299.2	3.87
32	856.8	437.3	126.9	1421.0	41.6	840.6	430.3	134.4	1405.3	3.91
64	853.3	598.3	135.1	1586.0	59.4	837.1	590.1	140.7	1567.9	4.26
128	850.5	816.9	149.3	1816.7	70.2	830.6	790.6	160.4	1781.6	4.78

required by simulated annealing approach are thus excessive.

It can be seen that the proposed algorithm is over thousands times as fast as simmulated annealing methods. The costs generated by simulated annealing were generally slightly better, although the solutions obtaining by our proposed algorithm were never worse by more than 5% in terms of total costs.

As can be seen, the results produced by the proposed algorithm decreased with each allocation cost. This improvement can be explained by observing the reassignment process of the proposed algorithm(i.e. Step 6:)

4.2 Conclusions

In order to fully utilize processing power it is advantageous to assign the program modules of a distributed program to the processors in such a way that the execution time of the entire program is minimized. The overloaded processors can perform its own processing with the performance degradation. Thus, the processor allocation of program modules is to be carried out so that each module is assigned to a processor whose capabilities are most appropriate for the module, and total cost is minimized that is sum of IPC cost and execution cost and imbalance cost of the assignment.

In this paper, we present an effiecient heuristic task allocation algorithm. The algorithm converges fast and obtains a good solution. Experimental results indicate that the proposed algorithm performs quite well on a variety of task-processor system. Heuristic algorithms do not always provide the optimum solution. However, in most of the cases, the algorithm presented in this paper does provide solutions that are close to optimum. Thus, the proposed algorithm is clearly preferable in a practical context.

참 고 문 헌

[1] R. K. Arora and S. P. Rana, "Heuristic Me-thods for Process Assignment in Distributed Computing Systems," Information Process-ing Lett., Vol. 11, pp.199-203, Dec. 1980.

[2] A. Barak and A. Shiloh, "A distributed load-balancing policy for a multicomputer," Software-Practice and Experience, Vol. 15, No. 9, pp. 901-913, Sept. 1985.

[3] S. H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," IEEE Trans. Software Eng., Vol. SE-5, pp. 341-349, July 1979.

[4] S. H. Bokhari, "the mapping problem," IEEE Trans. Comput., Vol. C-30, pp. 207-214, Mar. 1981.

[5] S. H. Bokhari, "Shortest Tree Algorithm for Optimal Assignments Across Space and Time in Distributed Processor Systems," IEEE Trans. Software Eng., Vol. SE-7, pp. 583-589, Nov. 1981.

[6] Nicholas S. Bowen, Christos N. Nikolaou, and Arif Ghafoor, "the Assignment Problem of Arbitrary Process Systems to Heterogeneous Distributed Computer Systems," IEE E Trans. Computer, Vol. 41, No. 3, pp. 25 7-273, MaRCH 1992.

[7] C. H. Lee, D. Lee, and M. Kim, "Task Assignment in Linear Array Networks," IEEE Trans. Computer, Vol. 41, No. 7, pp. 877-880, July 1992.

[8] K. R. Seo, C. H. Lee, and K. H. Park, "An efficient Task Allocation Algorithm for Homogeneous Tree Structure," 정보과학회 논문지, Vol. 20, No. 2, pp. 254-263, Feb. 1993.

[9] K. R Seo, K. H. Park, "Task Assignment in Host-satellite Systems," Journal of Circuits, Systems, and Computers, Vol. 6, No. 3, pp. 213-225, 1996.

[10] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," IEEE Computer, pp. 57-69, Nov. 1980.

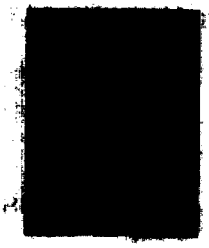
[11] K. Efe, "Heuristic Models of Task Assignm -ent Scheduling in Distributed Systems," IEEE Computer, pp. 50-56, June 1982.

[12] V. M. Lo, "Algorithms for Task Assignment in Distributed Systems," IEEE Trans. Computer, Vol. c-37, pp. 1384-1397, Nov. 1988.

[13] A.K. Sarje, and G. Sagar "Heuristic model for task allocation in distributed computer systems," IEE Proceedings-E, Vol. 138, No. 5, pp. 313-318, Sept. 1991.

[14] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Trans. Software Eng., Vol. SE-3, pp. 85-93, Jan. 1977.

[15] D. F. Towsley, "Allocating programs containing branches and loops within a multiple pro cessor system," IEEE Trans. Software Eng., Vol. SE-12, pp. 1018-1024, Oct. 1986.



서 경 룡

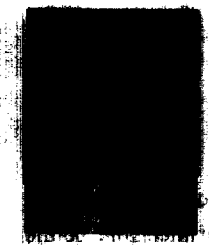
1983년 부산대학교 전기기계공학과 졸업(학사)

1990년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)

1995년 한국과학 기술원 전기 및 전자공학과 졸업(공학박사)

1992년~현재 국립부경대학교 컴퓨터공학과 조교수

관심분야 : 병렬, 분산시스템, 컴퓨터 네트워크



여 정 모

1980년 동아대학교 전자공학과 졸업(학사)

1982년 부산대학교 전자공학과 졸업(공학석사)

1988년 울산대학교 전자 및 전산 기공학과(공학박사)

1988년~현재 국립부경대학교 전자계산학과 부교수

관심분야 : CAD, 디지털시스템, 컴퓨터구조