IEEE Region 10 Conference on Computer and Communication Systems, September 1990, Hong Kong

Task Assignment Scheduling by Simulated Annealing

Feng-Tse Lin and Ching-Chi Hsu

Dept. of Computer Science and Information Engineering
National Taiwan University
Taipei , Taiwan

Abstract : The stochastic, heuristic search algorithm called simulated annealing is considered for the problem of static task assignment scheduling in distributed computing systems. The purposes of task assignment scheduling are to assign modules of programs over a set of interconnected processors in order to reduce the job turnaround time as well as to obtain the best system performance. In this paper, we show that the approach of simulated annealing, with a properly designed annealing schedule and a good move generation strategy, can be used to solve this problem in an efficient way.

1. Introduction

Distributed computing systems are made up of a set of processors that can be formed a loosely-coupled style multiprocessor. The task assignment scheduling is one of the fundamental aspects of distributed processing environments such that it deals with the question of optimally assigning modules to processors so as to minimize the cost of running a program. There is a problem of 'saturation effect' [3] caused by excessive interprocessor communications degrades the throughput of the system seriously. Consequently, the goals of task assignment scheduling are to minimize interprocessor communication cost and to maximize utilization of processors.

Unfortunately, there is always a conflict between the desire to minimize total execution cost and the desire to achieve load balancing which is to utilize the concurrency offered by the multiprocessor system. The general task assignment scheduling problem with a set of constraints, such as memory capacity, and processor capability, is found to be NP-hard[6]. Thus, a good algorithm for its optimal solution in polynomial time is very unlikely exist. For this reason, many researchers focus on development of suboptimal solutions of its general problems or optimal solutions of its special problems. Several approaches to the task assignment problems with various constraints to obtain optimal or near-optimal solutions have been suggested over the past ten years. They can be roughly classified into five categories, namely, graph theoretic [2,13], mathematical programming [3,7], heuristics [11], solution space enumeration and search [10] and dynamic probabilistic methods [12].

In this paper, we present another approximate approach for task assignment scheduling by simulated annealing. Over the past six years, theoretical studies of simulated annealing based on the equilibrium theory of Markov chains have shown that a global optimum state can be reached with probability one after an infinite number of transitions [6,8], and its applications to a broad class of combinatorial optimization problems in diverse areas are increased significantly [1,5]. Many researchers are quite agree that simulated annealing to be considered a powerful optimization tool is in particularly of no such efficient tailored algorithms are available for those NP-hard problems [5].

However, the main drawback of this approach is its long computation time required to converge to the globally optimal solution. Instead of finding the optimal solution, a straightforward approach to speed up the algorithm within acceptable computation time is to design of an efficient or parallel annealing schedule for finding a near-optimal solution. In this paper, we design an efficient annealing schedule for finding good approximate solution for task assignment scheduling in a significantly huge reduction of move iterations.

2. Task Assignment Scheduling Problem

Assuming that there are M modules $m_1,m_2 ...,m_m$ in a given task and N processors $p_1,p_2 ...,p_n$ in the system. An assignment of tasks to processors can be described formally as a function that maps the set of modules M to the set of processors P on which modules may be executed. A task assignment vector A is a function of $A : M \rightarrow P$, where $A(i) = j$ if module m_i is assigned to processor p_j, $1 \le i \le m$, $1 \le j \le n$.

2.1. The total cost of an assignment

(1) Execution cost

Let E be an m*n matrix representing the execution cost of a task in the system, where execution cost e_{ij} depends on the work to be performed by module m_i as well as on the attributes of processor p_j. The total execution cost of an assignment A is given by

$$COST1(A) = \sum_{i=1}^{m} e_{i,A(i)} .$$

(2) Communication cost

Let C be an m*m matrix representing the communication cost between modules in a task, where $c_{ij} = k$, if module m_i communicates with module m_j for some cost k and $A(i) \neq A(j)$. That is, any two modules that communicate during their execution incur a penalty if they are executed on different processors. The total communication cost of a given assignment A is given by

$$COST2(A) = \sum_{i=1}^{m} \sum_{j=i+1}^{m} c_{A(i),A(j)} \quad \text{for } A(i) \neq A(j).$$

Thus, the total cost of an assignment A is

$$COST(A) = COST1(A) + COST2(A).$$

2.2. Bottleneck in the system

In case of considering the processor with the heaviest loads that causes the bottleneck in the system, we need to find out the workload of each processor. The workload of the processor k is defined as $COST(A)_k$ with

$$COST1(A)_k = \sum_{i=1}^{m} e_{i,A(i)} \quad \text{for each } A(i) = k.$$

$$COST2(A)_k = \sum_{i=1}^{m} \sum_{j=i+1}^{m} c_{A(i),A(j)} \quad \text{for each } A(i) = k \text{ and } A(j) \neq k.$$

Then, the bottleneck in the system is the critical processor which has the maximum total cost. That is,

$CRITICAL(A) = \text{maximize } COST(A)_k \quad \text{for } 1 \leq k \leq n.$

2.3. The Constraints

(1) Memory capacities

The limited memory capacities in the system are represented as

$$\sum_{i=1}^{m} b_i \leq q_j \quad \text{for each } A(i) = j,$$

where b_i denotes the minimum amount of memory storage needed for module m_i, and q_j in vector Q represents the maximum memory capacity at processor p_j.

(2) Real-time deadlines

The real-time deadlines in a given task are represented as

$$\sum_{i=1}^{m} d_i \leq l_j \quad \text{for each } A(i) = j,$$

where d_i denotes the minimum processing time required by module m_i, and l_j in vector L denotes the time limit for processing the modules that executed on processor p_j for a given task. It states that the length of time required to process the modules assigned to a single processor must not exceed the required real-time deadline for that processor in application environments.

(3) Topology of the system

Let Tp be an n×n matrix representing the connection of processors in the system, where

$tp_{ij} = 1$, if processor p_i and p_j are connected each other;
0, otherwise.

2.4. The objective function

The total cost of an assignment in the system is equal to the maximum workload of the critical processor. Now, all we want to do are looking for an optimal assignment A which has the minimum cost in all of the possible assignments, i.e.

Minimize CRITICAL(A) for all possible A,

Subject to the constraints with *memory capacities, real-time deadlines and the system topology.*

3. Simulated Annealing

As early as 1953, Metropolis et al. proposed an algorithm for the efficient simulation of the evolution of a solid to thermal equilibrium. Kirkpatrick, Gelatt and Vecchi [4] realized that there exists a profound analogy between minimizing the cost function of a combinatorial optimization problem with many variables and the slow cooling of a solid until it reaches its low energy ground state. At each temperature T, the solid is allowed to reach thermal equilibrium, characterized by a probability of being in a state with energy E given by the Boltzmann distribution [1] of

$$Pr(E) = \frac{1}{Z(T)} \exp\left(-\frac{E}{K_b T}\right),$$

where Z(T) is the partition function and K_b is the Boltzmann constant. As the temperature decreases, the Boltzmann distribution concentrates on the states with lower energy states and finally, when the temperature approaches zero, only the minimum energy states have a non-zero probability of occurrence. The application of this method to a specific optimization problem requires careful design of the following ingredients.

(1) defining a formal description of the states of the system.
(2) formulating the cost function and /or the constraints of the system.
(3) designing a good move generation strategy to migrate a state to another.
(4) defining a good annealing schedule including,
  (a) the initial value of all parameters,
  (b) a rule for changing the current value of the control parameter into next one,
  (c) the number of repetitions at each value of the control parameter,
  (d) the condition of thermal equilibrium, and
  (e) the stopping criteria.

3.1. Configuration of the system

(1) Input data structures

A matrix E(1:m, 1:n) where e_{ij} denotes the execution cost of module m_i executed on processor p_j. A matrix C(1:m,

1:m) where c_{ij} denotes the communication cost between module m_i and m_j while executed on different processors. A matrix T(1:n,1:n) where t_{ij} denotes the connection information about processors p_i and p_j. A vector B(1:m) where b_i denotes the minimum storage needed for module m_i as well as a vector Q(1:n) where q_j denotes the maximum capacity of local memory in processor p_j. A vector D(1:m) where d_i denotes the minimum processing time required by module m_i as well as a vector L(1:n) where l_j denotes the real-time deadline for processor p_j in a given task.

(2) Output data structure

An assignment vector A(1 :m) where A(i) denotes which processors currently being assigned to module i.

3.2. Cost function

The objective function is to minimize {CRITICAL(A) + P(A)}, for all possible assignments of A. P(A) is a penalty function, $P(A) \neq 0$, if an assignment A violates the constraint of memory capacities or real-time deadlines, otherwise P(A) = 0. CRITICAL(A) is the workload of the critical processor in the system as defined in section 2. We define the cost function of an assignment A is F(A) = {CRITICAL(A) + P(A)}.

3.3. Move generation strategies

Consider the system with m modules and n processors, there are n^m possible states (assignments). For each state i, the neighbors of i is defined as generating from i by one step transition or by a small pertubation. Assuming that the neighborhood structure of the system is well-defined, for any node i the probability of to be visiting is bounded below by a small positive fraction, say p_k, at temperature T_k. Then, the probability of not visiting is bounded above by 1 - p_k. In the course of the annealing processes, it is easily to prove that each state in a well-defined neighborhood structure of the system after an infinity number of transitions is recurrent (i.e. visiting infinity often in time).

The simulated annealing process can be formulated as a sequence of homogenerous Markov chains, each chain is consisting of a sequence of states where the transition probability is defined. We have three strategies to use for generating the next state of the system. In short, we start with strategy 1 in an attempt to decrease the objective function. If it is not in this case, stategy 2 is attempted. Strategy 2 involves an exchange the roles of two objects such that we wish to decrease the cost of the system depend on the minimization of the objective function. The detrimental swaps are accepted according to the Metropolis's acceptance criterion. Strategy 3 is keeping the algorithm from getting trapped at a locally optimal by permitting an unexpected state with a higher cost.

An illustrative example is shown in **Fig. 1**. Assuming that we have eight modules and six processors in the system with the current assignment A = (1, 4, 3, 1, 5, 6, 2, 4). That is, modules 1 and 4 are assigned to p_1, modules 7 is assigned to p_2, module 3 is assigned to p_3, module 2 and 8 are assigned to p_4, module 5 is assigned to p_5, and module 6 is assigned to p_6. By strategy 1, module 4 is selected and reassigned to p_6. By startegy 2, modules 2 and 5 are selected and to interchange their assignments. Finally, by strategy 3, module 5 is assigned to p_4.

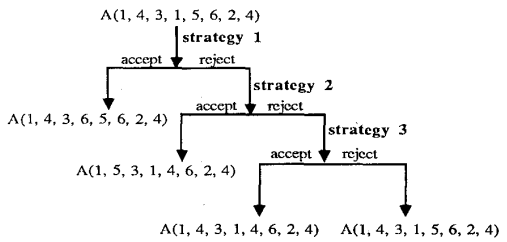


Fig. 1. Move generation strategies.

3.4. Annealing schedule

(1) Initial value of temperature (T_1)

The most straight way to consider a reasonable initial value of temperature T_1 is to be satisfied with min(1, exp(-ΔC / T_1)) = 1, for each possible ΔC in a given number of iterations. It is called the hot enough condition. From our considerations, the initial value of temperature should depend on the acceptable range of costs. Assuming that the acceptable costs over a large number of iterations is Gaussian distribution with mean μ and standard deviation σ. By using the central limit theorem and the law of large numbers, one can shown that the expected cost (μ) and the expected square cost (δ) can be approximated by the average of the cost function F(A) and the average of the square of the cost function $F^2(A)$, respectively, which are all sampled over R iterations. That is,

$$\mu = \frac{1}{R}\sum_{i=1}^{R} F(A), \text{ and}$$

$$\delta = \frac{1}{R}\sum_{i=I}^{R} F^2(A).$$

Such that, one can easily obtain the approximation of σ as

$$\sigma = \sqrt{\delta - \mu^2}$$

Let the maximum cost and the minimum cost accepted over R iterations be denoted by C_{max} and C_{min}, respectively. The hot enough condition is the inequality of

$$\mu - 2\sigma \leq C_{min} < C_{max} \leq \mu + 2\sigma .$$

We start with T_1 = max c_{ij} + max e_{ij} for each element c_{ij} in matrix C and each element e_{ij} in matrix E, and follow by performing a number of moves. If the current value of T_1 is not satisfied with the inequality we try to double it up.

(2) The number of iterations at each temperature

According to Markov chain theory, every irreducible and aperiodic Markov chain possesses a unique stationary distribution. The stationary distribution of a homogeneous Markov chain is the invariant probability distribution of the states after an infinite number of moves. However, we can

predict the long-run behavior of the system just over a short period of time by the characteristics of states in Markov chains. Let π_i be the invariant probability of being in state i, and w_{ii} be the transition probability for staying in state i. The mean recurrence time of state i is defined as

$$\mu_{ii} = \frac{1}{\pi_i},$$

and the expected number of transitions to leave state i is given by

$$\mu_{ij} = \frac{1}{1-\pi_i} = \frac{1}{1-w_{ii}}.$$

Nevertheless, we can approximately estimate μ_{ij} at each temperature T_k is bounded above by the following equation of

$$\mu_{ij} = \frac{1}{\sum_{j \in N(i)} w_{ij}(T_k)} \cong \frac{1}{\exp(-(C_{max} - C_{min})/T_k)},$$

and it is the expected number of transitions to escape from any local minimum.

Thus, for each value of T_k, the kth Markov chain is generated by performing L_k iterations. At the end of each L_k iterations, we make more μ_{ij} - L_k iterations if μ_{ij} - $L_k > 0$, otherwise calculate T_{k+1} and L_{k+1} the next annealing cycle is then initiated. Nevertheless, as T_k approaches to zero, the probability of the transitions being accepted are sharply decreasing and thus one can notice that L_k becomes very large. Therefore, for the sake of the efficiency of our algorithm, an upperbound of L_k is given to avoid extremely long computation time for low values of T_k.

(3) The rule for decreasing the temperature

Let $b_{ji}(T_k)$ be the acceptance probability from state j to state i at temperature T_k. By Metropolis criterion, we obtain the properties of the acceptance probability between states i and j with their costs F(i) < F(j) are $b_{ji}(T_k) = 1$ for all T_k, $b_{ij}(T_k) \uparrow 1$ as $T_k \rightarrow \infty$, and $b_{ij}(T_k) \downarrow 0$ as $T_k \rightarrow 0$. It is obviously that there must be exist $b_{ij}(T_k) = \beta * b_{ji}(T_k)$ and $0 \leq \beta \leq 1$. The asymptotic convergence of the algorithm needs that the detailed balance equation will be preserved. That is to say, in an irreducible Markov chain, for any two adjacent nodes i and j, the net number of moves from i to j is equal to the net number of moves from j to i. If the temperature drops too sharply, the detailed balance equation will not be preserved. Let $f_{ij}(T_k) = b_{ij}(T_k) / b_{ji}(T_k)$ be defined as the ratio of the probability of move from i to j to the probability of move from j to i at temperature T_k. Since $b_{ji}(T_k) = 1$ for all T_k, it is easily to prove that $f_{ij}(T_k) = b_{ij}(T_k) = \beta$, $0 \leq f_{ij}(T_k) \leq 1$, $f_{ji}(T_k) = 1 / b_{ij}(T_k) = 1 / \beta$, $1 \leq f_{ji}(T_k) \leq \infty$, and $f_{ij}(T_k) * f_{ji}(T_k) = 1$. Thus, the decrement ratio of T_k should keep β be monotonically piecewise decreasing in the interval of [0, 1].

(4) The stopping criterion

The stopping criteria determining the termination of simulated annealing algorithm. As the value of temperature is decreased, the likelihood of accepting moves that increases the cost function diminishes. If no more moves will be accepted in a given temperature, this tends to approach an optimum cost, the stopping criterion is then existing. In our considerations, the annealing curve is used for determining more efficient stopping criterion.

The annealing curve is a curve of expected cost (μ) versus temperature (T) such that with an efficient annealing schedule, the expected cost should be decreased in a uniform manner. The slope of the annealing curve, also known as specific heat [1], is given by

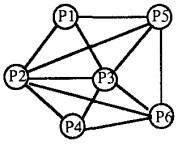
$$\frac{d\mu}{dT} = \frac{\sigma^2}{T^2}.$$

It is the rate of change of the expected cost with respect to temperature. When the freezing point is approaching, where the gross structure of the system are about to be fixed, the very careful annealing schedule is required. Beyond this freezing point, one can speeds up the cooling ratio through the low temperature until no further improvements are obtained. With the specific heat and critical temperature as the frozen conditions, we can substantially reduce a large number of redundant iterations at low temperature regions.

4. Experimental Results

In our implementation, we let $L_{k+1} = L_k * \theta$, where θ = 1.1, and L_k is ceiled by an upperbound ; the decrement rule is given by $T_{k+1} = T_k * \lambda$, where $\lambda = 0.8$. The initial number of iterations and the upperbound of iterations are depending on the number of modules and the number of processors of the system which are given by $L_1 = m*(m+n)$ and upperbound = $(m+n)*L_1$, respectively, where m is the number of modules of a given task and n is the number of processors in the system. On our experiments, we have four tasks in a six-processor multiprocessor system with their size and the complexity in increasing order. That is, tasks 1, 2, 3, and 4 are consist of six, eight, ten, and twelve modules, respectively. The data of the topology of the system, the execution costs and the communication costs of modules, the maximum capacity of each local memory, the expected processing time of each module, and the real-time deadlines for each processor are shown in **Table 1**.

The experimental results are shown in **Table 2**, where tasks 1 and 3 obtain optimal solutions, and tasks 2 and 4 obtain near-optimal solutions by the proposed annealing schedule. The table show that, for example, the final assignment of task 1 is (5,5,2,1,2,3) with the critical processor is P_1 and the cost is 161 at the total number of 5879 iterations and only 2320 assignments are accepted. We compare the performance obtained by the proposed annealing schedule with another conventional annealing schedule which are shown in **Fig. 2** and **Fig. 3**. The performance are the quality of solution and the running time required by the algorithms. The proposed annealing schedule always obtain optimal or near-optimal solution with the significant huge reduction of the number of iterations.



tologoly of the system

Task 1 : m1 - m6

Task 2 : m1 - m8

Task 3 : m1 - m10

Task 4 : m1 - m12

	P1	P2	P3	P4	P5	P6
m1	12	23	38	76	20	34
m2	45	15	30	25	16	21
m3	60	52	70	42	45	32
m4	45	55	60	80	62	51
m5	14	15	17	21	24	28
m6	32	18	26	29	37	24
m7	27	34	19	42	21	34
m8	41	23	53	29	42	24
m9	8	10	11	9	10	3
m10	14	20	32	16	15	18
m11	33	48	42	26	36	40
m12	56	43	37	34	32	41

execution costs -- matrix E

	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
m1	0	25	0	35	13	3	10	0	26	19	12	0
m2	25	0	18	0	0	21	0	14	0	35	0	3
m3	0	18	0	35	25	0	2	12	0	0	25	0
m4	35	0	35	0	12	34	0	0	0	19	0	0
m5	13	0	25	12	0	6	21	30	27	0	0	4
m6	3	21	0	34	6	0	15	0	0	0	11	0
m7	10	0	2	0	21	15	0	15	0	17	0	0
m8	0	14	12	0	30	0	15	0	0	24	15	0
m9	26	0	0	0	27	0	0	0	0	28	32	5
m10	19	35	0	19	0	0	17	24	28	0	0	0
m11	12	0	25	0	0	11	0	15	5	0	0	14
m12	0	3	0	0	4	0	0	0	5	0	14	0

communication costs -- matrix C

Vector B = (20,40,45,30,15,22,16,25,10,12,32,42)

Vector Q = (85,120,100,110,125,110)

Vector D = (5,4,3,1,2,5,4,6,1,2,3,4)

Vector L = (15,25,20,15,10,12)

Table 1. Data of four tasks for task assignment problems.

Task no.	Optimal Assignment	Final Assignment	Critical Processor	Iterations Total	Iterations Accepted
1 m=6	(5,5,2,1,2,3) cost = 161	(5,5,2,1,2,3) cost = 161	P_1	5879	2320
2 m=8	(1,3,5,1,2,3,2,2) cost = 189	(3,6,6,2,4,2,3,4) cost = 200	P_2	9328	3673
3 m=10	(2,4,4,3,6,3,6,6,2,2) cost = 263	(2,4,4,3,6,3,6,6,2,2) cost = 263	P_4	25234	7814
4 m=12	(2,2,4,3,6,3,6,6,4,2,4,3) cost = 307	(2,2,6,3,4,3,4,4,6,2,6,3) cost = 306	P_3	42765	13762

Table 2. Tabulation of experimental results of four tasks.

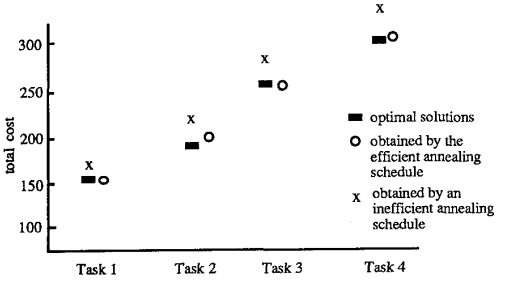


Fig. 2 Comparison between efficient and inefficient annealing schedule.

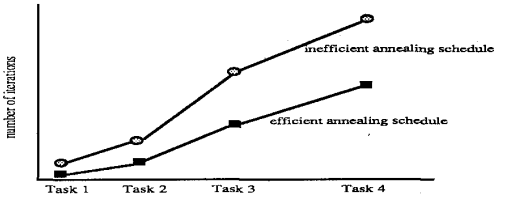


Fig. 3. The reduction of the number of iterations.

5. Conclusions

In this paper, we have presented an approximate approach by simulated annealing for task assignment scheduling problem. The performance of simulated annealing is heavily depends on the problem size as well as the range of all parameters to be given in the annealing schedule. The theoretical studies and the empirical rules are both critical to design a good annealing schedule. However, it is quite a different approach in comparisons with other proposed methods for task assignment scheduling problem. This approach allows other system constraints to be easily incoporated. With proposed efficient annealing schedule, the optimal solution or the near-optimal solutions can be obtained in a reasonable computation time. Further researches can be directed to parallel or distributed implementation of simulated annealing algorithm.

References

[1] **E. Aarts, J. Korst**, *Simulated annealing and Boltzmann machines - A stochastic approach to combinatorial optimization and neural computing*, John Wiley and Sons publishing, 1989.

[2] **S. H. Bokhari**, *Assignment problems in parallel and distributed computing*, Kluwer Academic Publishers, 1987.

[3] **W. W. Chu, L. J. Holloway, M.T. Lan, Kemal Efe**, "*Task allocation in distributed data processing*", Computer, vol. 13, Nov. 1980, pp 57-69.

[4] **S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi**, "*Optimization by simulated annealing*", Science, vol. 220, no. 4598, May 1983, pp 671-680.

[5] **Van Laarhoven, P. J. M. Aarts**, *Simulated annealing : theory and applications*, D. Reidel Publishing Company, 1987.

[6] **V. M. Lo**, "*Heuristic algorithms for task assignment in distributed systems*", IEEE Trans on Computers, vol. 37, no. 11, 1988, pp 1384-1397.

[7] **P. R. Ma, E. Y. S. Lee, M. Tsuchiya**, "*A task allocation model for distributed computing systems*", IEEE Trans on Computers , vol. C-31, Jan. 1982, pp 41-47.

[8] **F. Romeo, A. Sangiovanni-Vincentelli, C. Sechen**, "*Probabilistic hill climbing algorithms: Propreties and Applications*", Proc. 1985 Chapel Hill Conference on VLSI, May 1985, pp 393-417.

[9] **G. Sasaki, B. Hajek**, "*The time complexity of maximum matching by simulated annealing*", JACM, vol. 35, no. 2, Apr. 1988, pp 387-403.

[10] **C. C. Shen, W. H. Tsai**, "*A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion*", IEEE Trans. Computers, vol. C-34 , no. 3, Mar 1985, pp 197-203.

[11] **J. B. Sinclair**, "*Efficient computation of optimal assignments for distributed tasks*", Journal of parallel and distributed computing, Apr. 1987, pp 342-362.

[12] **J. A. Stankovic**, "*An application of Bayesian decision t heory to decentralized control of job scheduling*", IEEE Trans. Comput., vol. C-34, no. 2, Feb. 1985, pp 117-130.

[13] **H. S. Stone**, "*Multiprocessors scheduling with the aid of network flow algorithms*", IEEE Trans. Software Eng. , vol. SE-3, Jan. 1977, pp 85-93.