アルゴリズム 23－3
(1991. 9. 19)

ＯＰＳ５プロダクションシステムにおけるマッチ処理の並列化

川口 剛　本田 悦朗　増山 博

（宮崎大学工学部）

あらまし　プロダクションシステムにおける各処理の中で、マッチ処理は全実行時間の大半を占める。ＯＰＳ５では、ルールをネットワークに構造化することによってマッチ処理の高速化をはかっており、この手法はＲｅｔｅアルゴリズムと呼ばれる。本論文では、Ｒｅｔｅアルゴリズムの並列化手法を提案する。この手法の中では、Ｒｅｔｅネットワークのノードをプロセッサに静的に配置するための手法が工夫されている。

A PARALLEL MATCH ALGORITHM FOR OPS5 PRODUCTION SYSTEMS

Tsuyoshi Kawaguchi　Etsuro Honda　Hiroshi Masuyama
Faculty of Engineering, Miyazaki University

Abstract. The most time consuming step in the execution of production systems is the match step. The match algorithm used in OPS5 uses a special kind of data-flow network, called Rete network. In this paper we present a scheme for allocating the nodes of a Rete network onto processor $P_1,\cdots,P_m$. The scheme finds an allocation minimizing the maximum of $\pi(P_i)$, $1\leq i\leq m$, where $\pi(P_i)$ denotes the probability that more than one nodes assigned to P_i are simultaneously activated by the same working memory change. When hash tables are used for storing tokens on memory nodes, minimizing the maximum of $\pi(P_i)$, $1\leq i\leq m$, approximately leads to the minimization of the execution time of a match cycle.

1. Introduction

OPS5 is the most popular language for programming production systems [1]. OPS5 production systems perform the inference by cycling over three steps - match, conflict-resolution and act.

It is known that current OPS5 production system interpreters spend almost 90% of their execution time in the match step [2]. Thus, parallelism within the match step is especially important for the speed up of the execution of a production system although we can use parallelism within each of the three steps. The match algorithm used in OPS5 is called Rete and the algorithm uses a special kind of a data-flow network compiled from the left hand sides of rules [3]. And so, in OPS5, parallelism within the match step means a parallel implementation of the Rete algorithm.

In this paper we study the implementation of the Rete algorithm on general purpose multiprocessor systems which include both the shared memory multiprocessor systems and the message passing multiprocessor systems. Especially, for the message passing architectures, memory nodes of the Rete network need to be statically allocated onto multiprocessors before starting the execution of the production system although activations of two-input nodes can be dynamically scheduled onto multiprocessors during the execution [4].

Oflazer [5] and Dixit et al.[6] proposed schemes for statically allocating production (or rules) of a production system onto multiprocessors. However, as indicated by Gupta[2], the speedup obtained by node-level parallelism is much larger than that obtained by production-level parallelism.

The node-level parallelism requires a scheme for scheduling nodes of the Rete network onto

multiprocessors so as to minimize the execution time of the match step in each cycle. Takeda et al.[7] proposed a static node allocation scheme. After assigning weights to nodes, the scheme schedules the nodes onto processors so as to balance the total weight of the nodes assigned to each processor. However, what procedure was used for weighting the nodes is not clear from their paper.

In this paper we present a new scheme for static node allocation. After dividing a given set of processors into two subsets to perform separately the allocation of constant-test nodes and the allocation of two-input nodes, the scheme does the followings for each of these allocation steps.

The scheme first gives weights p(v,w) to all node-pairs v and w where the weights p(v,w) are computed by using a probabilistic model of match cycles. (More generally, p(v,w) can be viewed as a penalty cost incurred by assigning v and w to the same processor.)

Next, the scheme divides the nodes into several blocks $B_1,\cdots,B_m$ so as to minimize the maximum of $\pi(B_k)$, $1\leq k\leq m$, where $\pi(B_k)$ denotes the sum of p(v,w) over all node-pairs v and w assigned to block B_k. $\pi(B_k)$, $1\leq k\leq m$, approximates the probability that more than one nodes assigned to B_k are simultaneously activated by the same working memory change.

When hash tables are used for storing tokens on memory nodes, minimizing maximum of $\pi(B_k)$ approximately leads to the minimization of the execution time of each match cycle.

2. A Brief Overview of OPS5 Production Systems

2.1 OPS5

An OPS5 production system is composed of the production memory (PM) and the working memory (WM). PM is a set of "if-then" rules (productions) and WM is a set of data-items, called working memory elements (WMEs).

The WMEs are divided into several classes $C_1,\cdots,C_m$ and all WMEs of the same class have the same set of attributes. Each WME consists of the class name followed by pairs of attribute names and values. Fig.1 shows examples of WEMs.

The "if" part of a rule, which is called LHS, consists of a conjunction of condition elements (CEs). There are two types of CEs: the positive CEs and the negated CEs. A positive CE is a

t_1: (C_1 ^status= act ^ob=ladder)
t_2: (C_2 ^name=ladder ^on=floor)

Fig.1 Examples of WMEs

```
(P(C1 ^status=act ^ob [x])(C2 ^name=[x])
-->(modify1 ^ob=nil))

(P(C1 oper[x])(C2 ^weight=light ^hold=[x])
-->(make(C1 ^status=act ^oper=move)))
```

Fig.2 OPS5 rules

specification of restrictions on a WME and the CE matches a WME if all of the restrictions are satisfied by the values of the attributes of the WME. A negated CE, which is indicated by a minus sign(-), matches only if no WMEs match the corresponding non-negated CE. In a CE, an attribute value may be a variable. Although a variable can match any constant value, all occurrences of the variable must match identical values.

The "then" part of a rule, which is called RHS, consists of a set of actions. The RHS actions can make, remove or modify WMEs.

Fig.2 shows examples of OPS5 rules, where each variable is bracketed by [and]. For example, the first rule states the following:

if there are two WMEs t_1 and t_2 such that
(1) the class of t_1 is C_1 and t_1 has the value "act" for the "status" attribute,
(2) the class of t_2 is C_2,
(3) the "name" attribute value of t_2 equals the "ob" attribute value of t_1,
then change the "ob" attribute value of t_1 to "nil".

The inference engine of the production system cycles over three steps - match, conflict-resolution and act. The match step determines the set of rules that are matched by WMEs and inserts the instantiations of the rules into the conflict set. (When all CEs of a rule P are matched by a list of WMEs $(t_1,\cdots,t_m)$, the pair of P and $(t_1,\cdots,t_m)$ is called an instantiation of P.) The conflict-resolution step selects an

instantiation of a rule from the conflict set according to a selection strategy. The act step executes the RHS actions of the rule chosen in the conflict-resolution step.

2.2 The Rete Algorithm

The most time consuming step in the execution of production systems is the match step[2]. The match algorithm used in OPS5 is called Rete and the algorithm uses a special kind of a data-flow network compiled from the left-hand sides of rules[3]. The network for the rules of Fig.2 is shown in Fig.3.

At the beginning of each cycle, the changes to WM, which are called tokens, flow down the network from the root. There are two types of tokens: plus tokens and minus tokens. A plus token, which is generated by a make command, corresponds to the insertion of a WME. A minus token denotes the deletion of a WME and it is generated by a remove command.

There are four types of nodes in the network.

(1) Constant-test nodes: These are used to perform the constant tests of the CEs and always appear in the top part of the network. Specially, constant-test nodes which are the immediate successors of the root check the class names of tokens.

(2) Memory nodes: These store the tokens that match a part of the LHS of the associated rule. As can be seen in Fig.3, memory nodes appear on both inputs of a two-input node. If a plus token arrives at a memory node, it is simply sent to the successor two-input node. On the other hand, when a minus token enters into a memory node, the corresponding token is deleted from the memory node.

(3) Two-input nodes: Both inputs of a two-input node come from memory nodes. There are two types of two-input nodes: the and-nodes and the not-nodes. (While the and-nodes test the consistency of variable bindings between two positive CEs, the not-nodes implement the semantics of the negated CEs.) When a plus token

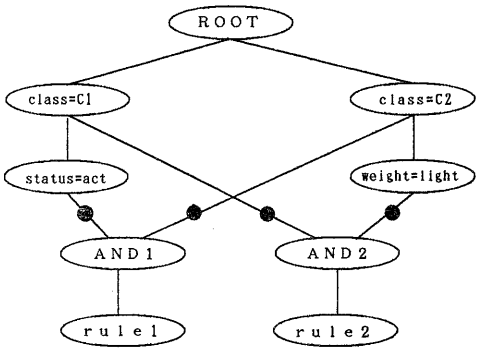


Fig.3 The Rete network for the rules of Fig.2

arrives at an and-node from the left memory or the right memory, it is compared to each token in the opposite memory for consistent binding of variables. If the compared token pair has consistent variable bindings, they are bound into a new token, called a composite token. On the other hand, a not-node checks whether or not an input token has inconsistent variable bindings with any token in the opposite memory. If so, a copy of the input token is sent to the successor node.

(4) Terminal nodes: Each terminal node corresponds to a rule in the OPS5 program. When a plus token flows into a terminal node, the pair of the token and the rule associated with this terminal node, which is called an instantiation, is inserted into the conflict set. On the other hand, if a minus token enters into a terminal node, the corresponding instantiation is deleted from the conflict set.

3. Data Structures Used in the Proposed Parallel Rete Algorithm

The parallel Rete algorithm proposed in this paper assumes the use of a multiprocessor system like that shown in reference [4]. The multiprocessor system consists of a control processor, conflict-set processors, constant-test node processors and two-input node processors.

Constant-test nodes and two-input nodes in the Rete network are divided into several parts by using the scheme proposed in the next section,

and these are placed on constant-test node processors and two-input node processors, respectively.

The control processor sends WM changes to the constant-test node processors. Further, WM changes that were successful in constant-tests are sent to the two-input node processors. The conflict-set processors perform conflict-resolution.

In the remainder of this section, we describe the data structures used in the proposed algorithm.

Let C_i, $1 \leq i \leq m$, denote class names used in a production system and let N_i, $1 \leq i \leq m$, be the number of attributes used in class C_i. Further, let A_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq N_i$, represent the attributes used in class C_i.

The following data structures are used in the proposed algorithm.

(1) Attribute values, which are character strings or constant numbers, are stored in a table, called a symbol table. The table is kept only by the control processor. Reference to the symbol table is not needed during the match step.

(2) Each WME of class C_i, $1 \leq i \leq m$, is represented by (N_i+1)-tuple $(x_0, x_1, \cdots, x_{N_i+1})$ where x_0 is the index of class C_i and x_j, $1 \leq j \leq N_i+1$, is the pointer to the value of attribute A_{ij} stored in the symbol table. The table storing WMEs is called a WME table.

(3) Each processor has a copy of the token table in its local memory. Each time a WME is inserted, the control processor sends copies of the WME to all processors.

(4) Composite tokens consisting of $t(\geq 2)$ WMEs are represented by t-tuple $(y_1, \cdots, y_t)$ where y_j, $1 \leq j \leq t$, denotes an address of the corresponding WME stored in the WM table. If T is a composite token to be stored in a memory node v and v is assigned to a processor P_k by using the node allocation procedure shown in the next section, then T is placed on only P_k.

(5) For match with LHSs of rules, the data shown in (2) and the data shown in (4) flow down the Rete network as a WM change (or a token) and as a composite token, respectively.

(6) Hash tables are used for storing tokens (or composite tokens) on memory nodes.

As indicated by Gupta [2], using hash tables, we can remarkably cut down the variance in processing times of two-input nodes.

Let T be the processing time of a two-input node activated by a token arrival and let t be the time required for a matching of a token-pair. When a hash table is used for each memory node, T can be approximated by $T=\max(st,t)$ where s denotes the number of successful matchings. Further Gupta[2] reported that the average values of s measured in six production systems were 1.41, 0.90, 1.06, 0.83, 0.60 and 0.71, respectively. Thus, we can assert that the average of T is nearly equal to t.

4. Static Allocation of Nodes onto Processors

In this section we describe a procedure for allocating the nodes of a Rete network onto multiprocessors.

For each two-input node v, let L(v) be the memory node on the left input of v and R(v) be the memory node on the right input of v. The proposed procedure places both L(v) and R(v) on the same processor as v. Further the procedure initially divides a given set of processors into two subsets S_1 and S_2 to allocate constant-test nodes on processors of S_1 and two-input nodes on processors of S_2.

We can use the same scheme for the allocation of constant-test nodes and for the allocation of two-input nodes. Therefore, in the remainder of this section, we will describe the scheme for the case when two-input nodes are allocated onto processors of S_2. The scheme uses a cost function derived from a probabilistic model of match cycles to evaluate the goodness of allocations. We first describe this model.

4.1 A Probabilistic Model of Match Cycles

The following notation is used.

P: a given OPS5 production system,

RN(P): Rete network compiled from the LHSs of the rules of P,

$\{C_1, \cdots, C_m\}$: set of class names used in P,

N_i, $1 \leq i \leq m$: the number of attributes of a WME included in class C_i,

A_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq N_i$: attributes of a WME included in class C_i,

n_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq N_i$: the number of values of A_{ij} taken by WMEs included in class C_i,

$a_{ij}(k)$, $1 \leq i \leq n_{ij}$: values of A_{ij} taken by WMEs included in class C_i.

Further we use the following assumptions.

(1) An input token to the Rete network, which corresponds to a WM change, is randomly selected from a population where tokens of class C_i, $1 \leq i \leq m$, distribute according to probabilistic density $q(C_i)$, respectively.

(2) If a randomized token is included in class C_i, $1 \leq i \leq m$, a value $a_{ij}(k)$, $1 \leq k \leq n_{ij}$, is found in attribute A_{ij} according to probabilistic density $q_{ij}(k)$, independently of other attributes.

Although we cannot estimate the exact values of $q(C_i)$ and $q_{ij}(k)$ before completing the execution of a production system, approximate values of these can be obtained through a test run of the production system. In the simulation experiments shown later, we roughly estimated $q(C_i)$ and $q_{ij}(k)$ as follows:

$$q(C_1)=\cdots=q(C_m)=1/m,$$
$$q_{ij}(1)=\cdots=q_{ij}(n_{ij})=1/n_{ij}.$$

And these experiments showed that even when such rough estimations are used for $q(C_i)$ and $q_{ij}(k)$, the proposed scheme gives an allocation much better than that obtained by a simple algorithm which, selecting nodes according to the left to right ordering, schedules these nodes onto processors so as to balance the number of nodes on each processor.

4.2 The Cost Function Used in The Proposed Scheme

For each pair of two-input nodes v and w in the Rete network, let p(v,w) denote the probability that a randomized token reaches both v and w. Further given a set of two-input nodes B, let $\pi(B)$ denote the sum of p(v,w) over all node-pairs v and w included in B. The proposed scheme partitions the set of two-input nodes into subsets B_i, $i \leq i \leq m$, to minimize

maximum of $\pi(B_i)$, $1 \leq i \leq m$.

Let N_i, $1 \leq i \leq m$, be a random variable which denotes the number of two-input nodes in B_i activated by a randomized token. If the probability that $N_i > 2$ is sufficiently small, $\pi(B_i)$ is nearly equal to the probability that $N_i > 1$.

If we can assume that nodes have similar processing times, the number of blocks m is set to the number of processors. Then, the minimization of maximum of $\pi(B_k)$ approximately leads to the minimization of the execution time of each match cycle. (The above assumption is valid for constant-test nodes. Further, in many production systems, the above assumption also holds for two-input nodes when hash tables are used for storing tokens on memory nodes[2].)

For the case when the variance in processing times of two-input nodes is not small, we set m to the average number of two-input nodes activated by a working memory change and assign the same number of processors to each block. Then, again, the minimization of maximum of $\pi(B_k)$ approximately leads to the minimization of the execution time of each match cycle.

4.3 Computation of p(v,w)

In this subsection, we will describe a method computing p(v,w) for each pair of two-input nodes v and w. For each two-input node v in the Rete network, let L(v) and R(v) be its left memory and its right memory, respectively. It is not possible that the same token arrives at both the left memory and right memory of a two-input node. Thus we have

$$p(v,w)=\sum_{x \in \{L(v),R(v)\}} \sum_{y \in \{L(w),R(w)\}} p(x,y)$$

where p(x,y), $x \in \{L(v),R(v)\}$ and $y \in \{L(w),R(w)\}$, denotes the probability that a randomized token reaches both x and y.

Further, p(x,y) can be computed as follows.

(i) If a predecessor of x is mutually exclusive to a predecessor of y, then p(x,y)=0. (Let E_x denote the event that a randomized token succeeds in the test performed on a constant-test node x. If $Pr(E_x \cap E_y)=0$, nodes x and y are said to be mutually exclusive.)

(ii) Further if a parent of x and/or a parent of y are two-input nodes, then p(x,y)=0.

(iii) Otherwise, let the path from the root to x be denoted by $x_0,x_1,\cdots,x_{t+1}$ ($t \geq 0$) where x_0 is the root and x_{t+1} is x, and let the path from the root to y be denoted by $x_0,x_1,\cdots,x_r,y_1,\cdots y_{s+1}$ ($0 \leq r \leq t, s \geq 0$) where y_{s+1} is y and x_r is the lowest common ancestor of x and y. Then we have:

$$p(x,y)=q(x_1)\times\cdots\times q(x_t)\times q(y_1)\times\cdots\times q(y_s)$$

where, for each node v, q(v) denotes the probability that a randomized token succeeds in the test performed on node v. q(v) can be computed by using the probability distribution of the attribute checked on node v.

As an example, we consider the network of Fig.4 where C_i ($1 \leq i \leq 7$), A_i ($1 \leq i \leq 6$) and P_i ($1 \leq i \leq 4$) denote constant-test nodes, and-nodes, and terminal nodes, respectively, and moreover L_i and R_i ($1 \leq i \leq 6$) represent the left-and the right-memories of A_i, respectively. The number put by the left side of each C_i denotes $q(C_i)$, which is the probability that a randomized token succeeds in the test performed on C_i. Further, we assume that $\{C_1,C_2,C_3\}$ and $\{C_4,C_5\}$ are the mutually exclusive node set in the network.

Then, $p(A_i,A_j)$, $1 \leq i \leq j \leq 6$, take the values show in Fig.5. As an example, we consider the computation of $p(A_1,A_3)$. Using (iii), we have first $p(L_1,L_3)=q(C_1)\cdot q(C_4)=1/9$ and $p(R_1,R_3)=q(C_2)\cdot q(C_6)=1/12$. Further, $p(L_1,R_3)=0$ because C_1 and C_2 which are predecessors of L_1 and R_3, respectively, are mutually exclusive. Similarly, we have $p(R_1,L_3)=0$ because the mutually exclusive node-pair, C_1 and C_2, are also predecessors of L_3 and R_1, respectively. Thus, $p(A_1,A_3)=p(L_1,L_3)+p(R_1,L_3)+p(L_1,R_3)+p(R_1,R_3)=7/36$

4.4 Finding an Optimal Partition

The problem introduced in the last subsection is formally stated as follows.

(*1) Given an integer number m and a weighted complete graph $G=(V,E)$ where $V=\{v_1,\cdots,v_n\}$ is its node set, E is its edge set and $p(v_i,v_j)$, $1 \leq i \leq j \leq n$, are weights of edges (v_i,v_j),respectively, partition V into m subset B_k, $1 \leq k \leq m$, so as to minimize the maximum of $\pi(B_k)$, $1 \leq k \leq m$.

The decision version of the above optimization problem is stated as follows.

(*2) For a real number K, does there exist a partition of V into m subsets B_k, $1 \leq k \leq m$, such that $\pi(B_k) \leq K$ for all B_k?

We can show that the graph coloring problem[8] which is a well known NP-hard problem is reduced to the above problem. Given an arbitrary graph $G'=(V',E')$ where $V'=\{v_1',\cdots,v_n'\}$ is its node set and E' is its edge set, let G denote a weighted compete graph with the node set $\{v_1,\cdots,v_n\}$ whose edges (v_i,v_j) have weights $p(v_i,v_j)$ defined as follows: $p(v_i,v_j)=1$ if $(v_i',v_j') \in E'$ and otherwise $p(v_i,v_j)=0$. Then, G' is m-colorable if and only if the node set of G can be partitioned into m subsets B_k, $1 \leq k \leq m$, in such a way that $\pi(B_k) \leq 0$ for all B_k.

In the remainder of this subsection,we present an optimization algorithm and an approximation algorithm for the problem (*1).

4.4.1 An Optimization Algorithm

The proposed optimization algorithm is similar to those shown in references [6] and [9]. The algorithm uses a branch-and-bound method.

Define variables x_{ij}, $1 \leq i \leq n$ and $1 \leq j \leq m$, as follows: $x_{ij}=1$ if node v_i is put in subset B_j, and otherwise $x_{ij}=0$. Then, the problem (*1) can be formulated as follows:

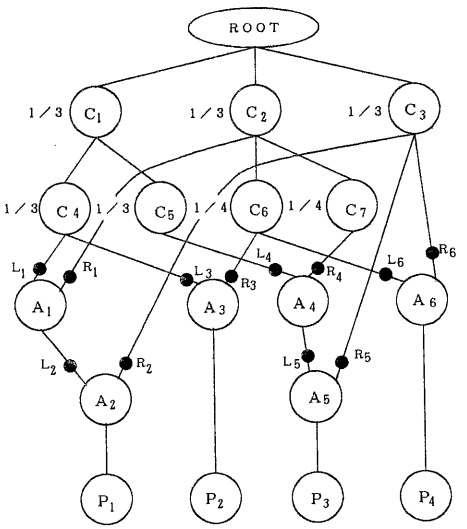


Fig.4 A probabilistic model of Rete network

A_i \ A_j	A_2	A_3	A_4	A_5	A_6
A_1	0	7/36	1/12	0	1/12
A_2		0	0	1/3	1/3
A_3			1/48	0	1/12
A_4				0	1/48
A_5					1/3

Fig.5 $p(A_i,A_j)$ for node-pair A_i and A_j in the network of Fig.4

minimize $\max_{1\le j\le m}\{\sum_{k=1}^{n-1}\sum_{i=k+1}^{n} x_{kj}x_{ij}p(k,i)\}$

subject to

$$\sum_{j=1}^{m} x_{ij}=1 \qquad \text{for } 1\le i\le n,$$

$$x_{ij}=0 \text{ or } 1 \qquad \text{for } 1\le i\le n \text{ and } 1\le j\le m.$$

Let all variables x_{ij} be initially free. Note that if a variable x_{ij} is fixed to one, variables x_{ik}, $1\le k\le m$ and $k\neq j$, need to be fixed to zero. Let Q denote the set of indices i, $1\le i\le n$, such that x_{ij}, $1\le j\le m$, are already fixed, and let $\bar{Q}=\{1,\cdots,n\}-Q$. Further, for each $i\in Q$, let J(i) denote the index j for which $x_{ij}=1$.

The search starts with the initial state in which all variables are free, and it proceeds by expanding the current state into new states. A state of the search space is represented by (Q,X) where X represents the set of pairs (i,J(i)), $i\in Q$. Further, a state (Q,X) is expanded into new states in the following way.

(i) We select an index $\alpha\in\bar{Q}$.
(ii) Let β be the maximum of J(i), $i\in Q$
(iii) if $\beta<m$, generate $(\beta+1)$ new states $(Q\cup\{\alpha\},X\cup\{(\alpha,j)\})$, $1\le j\le\beta+1$. Otherwise, generate m new states $(Q\cup\{\alpha\},X\cup\{(\alpha,j)\})$, $1\le j\le m$.

Associated with each state (Q,X), we compute a cost f(Q,X). For each j, $1\le j\le m$, let R(j) denote the set of indices $i\in Q$ such that $(i,j)\in X$, and let S_j represent the sum of p(k,i) over all pairs, k and i, included in R(j). Further, for each $i\in\bar{Q}=\{1,\cdots,n\}-Q$, let

$$C_i^*=\min_{1\le j\le m}\{\sum_{k\in R(j)} p(k,i)\}.$$

Then, f(Q,X) is defined by:

$$f(Q,X)=\max(g(Q,X),h(Q,X))$$

where

$$g(Q,X)=\max_{1\le j\le m}\{S_j\},$$

$$h(Q,X)=\frac{1}{m}\{\sum_{j=1}^{m} S_j+\sum_{i\in\bar{Q}} C_i^*\}.$$

A state (Q,X) is said to be a goal state if the cardinality of Q equals n. A goal state (Q,X) corresponds to a feasible solution for the problem (*1) and its cost f(Q,X) gives the objective function value associated with the solution. For a nongoal state (Q,X), let $f^*(Q,X)$ represent the minimum of $f(Q_i,X_i)$ over all goal states (Q_i,X_i) obtained by expanding (Q,X). Then, we have

$$f(Q,X)\le f^*(Q,X).$$

Therefore, if we obtain a goal state with cost f_0, states (Q,X) satisfying $f(Q,X)\ge f_0$ can be eliminated before their expansions.

4.4.2 A Heuristic Algorithm

Fig. 7 shows a heuristic algorithm for the problem (*1), which is called ALLOCATION. The algorithm uses a function called COLORING, which is shown in Fig. 6. Given a positive real number θ, the function COLORING returns "true" if it finds a partition of the node set $(B_1,\cdots,B_m)$ such that $\pi(B_k)\le\theta$ for all B_k, and otherwise it returns false. The time-complexity of this function is $O(n^2)$ where n denotes the number of nodes in a weighted complete graph.

Using a binary search method, the procedure ALLOCATION finds the minimum real number θ such that COLORING(θ) returns "true". The time-complexity of this procedure is $O(n^2\cdot\log(M/\varepsilon))$ where M and ε are the input parameters shown in Fig. 7.

5. Performance Evaluation

We performed simulation experiments to estimate the performance of the node allocation scheme described in the last section. In these experiments, we used the following assumptions: if the time needed by a test performed on a constant-test node is t, the time required for a matching of a token-pair on a two-input node is 3t and the time needed for the communication of a token is t/2.

The result of Fig.8 was obtained from the OPS5 production system for monkey and bananas[1]. The horizontal axis of this figure denotes the number of processors assigned to the two-input nodes. (The number of processors assigned to constant-test nodes was fixed to six.) The vertical axis of this figure represents the speedup rate RS(m) given by:

$$RS(m)=T(1)/T(m)$$

where T(1) is the execution time by uniprocessor and T(m) is the execution time by m processors.

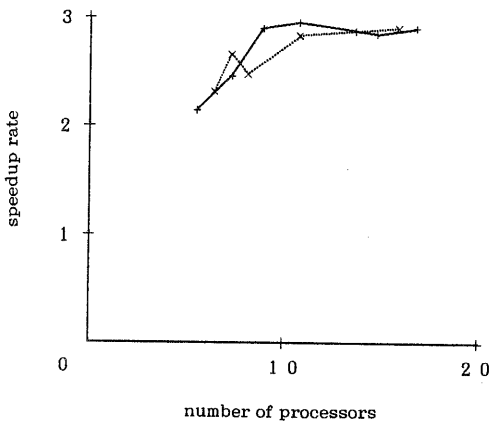
The solid line denotes the speedup rate

```
functoin COLORING(θ):
  begin
    make all Bi, 1 ≤ i ≤ m, empty;
    initialize U to the set of all two-input nodes;
    i := 0;
    while U ≒ φ and i ≤ m do begin
      for each v ∈ U do
        A(v) := Σ w∈U P(v,w);
      select v ∈ U with the greatest A(v);
      move v form U to Bi;
      π(Bi) := 0;
      while π(Bi) ≤ θ and U ≒ φ do begin
        for each w ∈ U do
          C(w) := Σ v∈Bi P(v,w);
        select w ∈ U with the smallest C(w),
        where tie is broken by
        selecting w with the greatest A(w);
        move w from U to Bi;
        π(Bi) := π(Bi)+C(w)
      end; {while}
      if U = φ then
        return(true)
      else
        if i = m then
          return(false)
        else begin
          let x be the last node added to Bi;
          move x from Bi to U;
          i := i+1;
    end;{while}
  end;
```

Fig.6 The function COLORING

```
procedure ALLOCATION:
  begin
    θL := 0;
    θU := M;
    while θU-θL> ε do begin
      θ := (θL+θU)/2;
      if COLORING(θ) then
        θU := θ;
      else
        θL := θ;
    end;
  end;
```

Fig.7 The procedure ALLOCATION



—— the proposed scheme
---- a simple node allocation scheme

Fig.8 Comparison of the performance between the proposed scheme and a simple node allocation scheme which assigns nodes to processors according to left to right ordering.

obtained by the scheme proposed in this paper. The dotted line denotes the speedup rate obtained by a simple scheme which, selecting two-input nodes according to the left to right ordering, assigns the same number of nodes to each processor. For example, if two-input nodes of the network shown in Fig. 4 are partitioned into three blocks by using this scheme, we have a partition $((A_1,A_2),(A_3,A_4),(A_5,A_6))$.

6. Conclusions

In this paper we have presented a scheme for allocating the nodes of a Rete network onto multiprocessor. The scheme finds a partition $(B_1,\cdots,B_m)$ minimizing the maximum of $\pi(B_i)$, $1\leq i\leq m$, where $\pi(B_i)$ is an approximation of the probability that more than one nodes of B_i are simultaneously activated by the same WM change. When hash tables are used on memory nodes, minimizing the maximum of $\pi(B_i)$, $1\leq i\leq m$, approximately leads to the minimization of the time needed by a match cycle.

References

[1] L.Brownston et al.: "Programming Expert Systems in OPS5", Addison-Wesley (1985).

[2] A.Gupta: "Parallelism in production systems",Morgan Kaufmann (1987).

[3] C.L.Forgy: "Rete: A fast algorithm for many pattern/many object pattern match problem", Artificial Intelligence, Vol.19, pp.17-37 (1982).

[4] A.Gupta and M.Tambe: "Suitability of message passing computers for implementing production systems", Proc. of AAAI'88, PP. 687-692.

[5] K.Oflazer: "Partitioning in parallel processing of production systems", Proc. of 1984 Int. Conf. on Parallel Processing, pp. 92-100.

[6] V.V.Dixit and D.I.Moldovan: "The allocation problem in parallel production systems", J. Parallel and Distributed Computing, Vol.8, pp.20-29 (1990).

[7] K.Takeda et al.: "A prallel implemention of the Rete algorithm", Proc. of 1989 Joint Symposium on Parallel Processing, pp57-64 (in Japanese).

[8] M.R.Garey and D.S.Johnson: "Computer and Intractability: A Guide to the Theory of NP-Completeness", Freeman, San Francisco (1979).

[9] C.C.Shen and W.H.Tsai: "A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion", IEEE Trans. on Computers, Vol. C-34, No.3, pp.197-203 (1985).