1 of 1

Data-Driven Parallel Production Systems: A Macro Actor/Token Implementation†

Andrew Sohn and Jean-Luc Gaudiot

Dept. of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-0781

Abstract

The importance of production systems in artificial intelligence has been repeatedly demonstrated by a number of expert systems. Much effort has therefore been expended on finding an efficient processing mechanism to process production systems. While data-flow principles of execution offer the promise of high programmability for numerical computations, we study here variable resolution actors, called *macro actors*, a processing mechanism for production systems. Characteristics existing in the production system paradigm are identified, based on which we introduce *macro tokens* as a companion to macro actors. A set of guideline is identified in the context of production systems to derive well-formed macro actors from primitive micro actors. Parallel pattern matching is written in macro actors/tokens to be executed on Macro Data-flow simulator. Simulation results demonstrate that the macro approach can be an efficient implementation of the production system paradigm.

1. Introduction

A major obstacle in the processing of artificial intelligence applications lies in the large search/match time. In rule-based production systems, for example, it is often the case that the rules and the database needed to represent a particular production system in a certain problem domain would be on the order of hundreds to thousands of rules and assertions. It is thus known that simply applying software techniques to the matching process would yield intolerable delays. Indeed, as Forgey [3] has pointed out, the time taken to match patterns over a set of rules can reach 90% of the total computation time spent in expert systems. The need for faster execution of production systems has spurred research in both the software and hardware domains [3,6,7,10,11].

From the software perspective, not only the matching step, but also parallel firing of the productions have been studied. The Rete match algorithm has been developed to utilize the temporal redundancy in the production systems [3]. Further optimization of the Rete algorithm has been studied in TREAT algorithm [10], which supports the conflict set. Parallelization of the Rete algorithm has been reported to suit the multiprocessor environment [6]. To speedup the processing of PS, parallel firing has been studied.

From the hardware perspectives, there have many stud-

ies reported, including message passing architecture [9] and shared memory multiprocessors [8]. The conventional control-flow model of execution is however limited by the "von Neumann bottleneck [2]." Architectures based on this model cannot easily deliver large amounts of parallelism [1]. The data-driven model of execution has therefore been proposed as a solution to these problems. The applicability of data-flow principles of execution to matching operations for production systems has been studied in [5].

In this paper, we further explore the applicability of data-flow principles of execution to production systems. Based on our observation that AI problems exhibit distinctive characteristics against numeric computations, we investigate in this paper an approach to obtain the medium grain parallelism, called macro actor/token.

We shall start our discussion in section 2 by introducing two fundamental approaches to AI processing. Section 3 finds some characteristics existing in production systems from the parallel processing perspective, based on which the macro data-flow principle is explained from AI perspective. A brief analysis is presented to show why medium grain macros are preferred to fine grain micros. Section 4 discusses several strategies about how to derive well-formed macros from micros for production systems. Section 5 gives a simulation result based on our execution model, macro data-flow simulator. Performance evaluation is also discussed in the section. Conclusions as well as future research on this study are made in the last section.

2. Parallel Processing of Production Systems

A production system (PS) consists of a *Production Memory* (PM), a *Working Memory* (WM), and an *Inference Engine* (IE). PM (or rulebase) is composed of *productions* (or rules), each of which performs predefined actions (right-hand side, RHS) if all the necessary conditions (left-hand side, LHS) are satisfied. The productions operate on WM which is a database of assertions, called *Working Memory Elements* (WMEs). The inference engine repeatedly executes an inference cycle which consists of three steps: *pattern matching*, *conflict resolution*, followed by *rule firing*. The inference engine halts either when there are no satisfied rules or when the solution is found.

From the parallel processing perspective the PS paradigm can be viewed as a composition of *local*- and *global latencies*. The local latency, t , is the processing time of an inference cycle in the PS paradigm. Each step in the production cycle is considered a local latency, as shown in Fig.1(a).

†This research was supported in part by the U.S. Department of Energy, Office of Energy Research under Grant No. DE-FG03-87ER25043.

MASTER

The global latency, T , depicted in Fig.1(b), is the processing time for searching a state space. Given an initial state, the inference engine finds next states by executing an inference cycle. Based on the control strategy or heuristics, the system decides which state in the search tree it will explore. The global latency, T , is thus linearly proportional to the number of states, n , to be explored in the search tree.

Techniques to reduce the global latency, T , in the PS paradigm can be classified basically into two categories: (1) *hardware/software parallel processing*, (2) *adaptive/heuristic processing* [14]. A straightforward technique would be to use as many number of Processing Elements (PEs) as needed. This would allow all branches to be explored in parallel as the search tree grows. This simple hardware approach with an infinite number of PEs can eliminate problems associated with backtracking and can hopefully find a desired solution in a finite amount of time. However, this technique is impractical and too costly since for most AI problems the number of possible states in the search tree would be exponential even for the problems of a modest size.

A way of reducing T from the adaptive/heuristic perspectives is to prune unpromising branches in the search tree by deriving *heuristics* at compile time from the problem domain (or learning them at run time) and applying them. We have investigated this second approach along neural networks and shall not consider further in this paper.

Fig.1: A search tree consisting of inference cycles. (a) an inference cycle before parallel processing, (b) after parallel processing. PM, CR, and RF stand respectively for pattern matching, conflict resolution, and rule firing

Our approach in this paper is centered around the data-flow principles of execution, more specifically, the *macro* data-flow principles [4]. As we shall see below, PSs exhibit some distinctive characteristics. One of such characteristics found in pattern matching is a list processing from which medium grain parallelism can be extracted. Based on these characteristics, we shall proceed with macro implementation of the PS paradigm.

3. Characteristics of Macros

There are two fundamental characteristics of the use of macros for processing AI problems: *macro actor* and *macro token*. A macro-actor is a collection of scalar instructions. The objective behind lumping instructions into one is to improve performance by exploiting locality in the instructions.

A macro-token is a collection of primitive data tokens. The basic data object in AI is a list. Consider an assertion $IS(X Y)$. This assertion, when implemented, can be represented as a list of three elements $(IS X Y)$. If we break it into three elements and form three data tokens (IS) , (X) , and (Y) as a basic element to operate on, each of these three tokens does not carry useful information.

When viewed from the architectural perspective, macros will substantially reduce the overhead in matching tags of data tokens. In dynamic data-flow principles, tokens carry tags which identify themselves from the context, code block, or instance of a loop they belong to. If the fact $(IS X Y)$ is split into three data tokens and is compared with another three data tokens (IS) , (X) , and (Z) , the tag matching time for three pairs of six data tokens is not less than three time units. However, when the two facts are compared in two lists, the tag matching time is *only 1!*

Fig.2: A data-flow graph in *micro-actors* for the comparison of two lists $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$. There are $n/2$ comparison actors to obtain a *maximum* parallelism.

Consider a typical match operation, shown in Fig.2, which compares two patterns, $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$. To achieve the maximum parallelism existing in the fine grain micro approach, the n -pairs can be simultaneously compared in n PEs, each of which is connected through $(\log n)$ -dim hypercube. Assume that two neighboring PEs are connected through three facilities (two communication nodes and a link) and that each PE has four facilities connected in a pipe, where each facility takes τ . The total time to process n comparisons on n PEs would be

$$\begin{aligned} t_{\text{micro},n} &= t_c + \lceil \log_2 n \rceil t_a + \lceil \log_2 n \rceil t_r \\ &= 4(1 + \lceil \log_2 n \rceil)\tau + 3\lceil \log_2 n \rceil\tau = (4 + 7\lceil \log_2 n \rceil)\tau, \end{aligned}$$

where t_c = comparison time, t_a = addition time, and t_r = routing time. In this simple algebra, no token waiting in the matching/store unit of each PE is taken into account. Furthermore all the comparison actors are *ideally* allocated to neighboring PEs (which is impractical in reality).

The total time to compare two lists on 1 PE for the macro approach is

$$t_{\text{macro},1} = nt_c + (n-1)t_a = 4(2n-1)\tau.$$

The ratio of the time taken for macros with 1 PE to mi-

cross with n PEs is

$$R = t_{\text{macro},1}/t_{\text{micro},n} = 4(2n-1)/(4+7\lceil \log_2 n \rceil) = O(n/\log_2 n).$$

Note that in the micro-actor approach, we assumed that the token routing would be done in 1 step, i.e., 3τ . In general, such a one-step routing is impractical for a 6-dim hypercube topology. Considering that the average number of elements in a pattern is five for production systems, the ratio is $R=5/(\log 5)=1.7$. The macros will outperform since there is no communication overhead involved in macros.

There are, however, drawbacks in using macros. If the primitive actors are grouped and formed into macros, the parallelism in fine grain processing will be lost. The grouping must be carefully made so as to avoid inefficient macros. Putting too many micros into a macro will apparently decrease the parallelism in fine grain processing, resulting in performance degradation. Forming a macro with too few micros will not give a noticeable improvement in performance. The PS paradigm which we are considering for our application domain provides a data parallelism existing in many patterns and WMEs. By putting too many micros into a macro, the data parallelism will likely diminish. The formation of macros is heavily dependent on problem domain. There must be a set of guidance criteria for the formation of macros. In the following section, we shall identify several rules from the PS paradigm and establish criteria to guide the grouping process, thereby producing efficient and well-formed macros.

4. Formation of Macros

Let A be a set of micro (or primitive) actors, $\{a_1, \dots, a_n\}$, and T_A be a set of data tokens, $\{t_1, \dots, t_m\}$, manipulated on A . Let B be a macro actor derived from A such that $B \subseteq A$. Let t_1 be a time taken to process a micro actor on a PE. Let t_n be a time taken to process A on n PEs. Let T_1 be a time taken to process a macro on a PE. Let r be a ratio of T_1 to t_n , i.e., $r=T_1/t_n$. A macro B is said to be *well-formed* if $r < \epsilon$, where $\epsilon \leq 2$.

An objective behind setting such a ratio is in the fact that if the processing time of the macro is not more than twice of the processing time for the corresponding micros in an ideal environment, we shall form a macro from micro actors. The ideal environment refers to the ideal allocation of micro actors on n PEs and ideal routing policy on data tokens. As we discussed earlier in section 3, achieving such an ideal environment would be impractical. The macros would outperform because of no data token routing, no waiting for other mating data token for two operand instructions, and so on. In this paper, we simply set ϵ to 2 for macro formation. We now briefly describe the formation of well-formed macros (*wfms*).

Let I_i be a set of input tokens to a_i and O_i be a set of output tokens from an actor a_i . We denote the dependence relation for $a_i, a_j \in A$ as follows: If $O_i \subseteq I_j$ such that $i \neq j$, $a_i \angle a_j$ for all i and j , where the dependence operator $\angle$ implies that a_i must be executed before a_j . By applying the dependence relation $a_i \angle a_j$ to A , we obtain an ordered set of actors,

$B = \{b_1, \dots, b_m\}$, where $b_i = \{a_i \mid a_i \angle a_j \text{ is not true}\}$. The dependence distance for $b_i, b_j \in B$ is defined as $d(b_i, b_j) = d_{i,j} = i - j$. The maximum dependency distance $d_{\max}$ for B is $m-1$.

We list below a set of guidelines for the formation of wfms and shall not prove the correctness due to space constraints:

1. (*Flow Dependency*) Let a_i and a_j be two actors. A macro $M = \{a_i, a_j\}$ is defined if $O_i \subseteq I_j$ and $d_{i,j} = 1$, where O_i is an output of a_i , I_j is an input to a_j , and $d_{i,j}$ is a dependence distance between a_i and a_j .
2. (*Encapsulation Effect*) Let a be a comparison actor and b be a set of true/false actors $\{b_1, \dots, b_n\}$. A macro $M = \{a, b\}$ is defined if $O_a \subseteq I_b$ and $d_{a,b} = 1$. This guide is based on the fact that the unnecessary true/false actors can be eliminated when the macro is formed because the formation of macro has an ability to output tokens.
3. (*List Processing*) Let $A = \{a_1, \dots, a_n\}$, and L be a list of m data tokens $\{t_1, \dots, t_m\}$. Let $I = I_1 \cup \dots \cup I_n$ and $O = O_1 \cup \dots \cup O_n$. A macro $M = \{a_1, \dots, a_n\}$ is defined if $I_i \subseteq I \cup O \cup L$ for $1 \leq i \leq n$ and $O_k \notin I$ for $k = d_{\max}$. The objective behind this guide is in preserving the semantics of the list.
4. (*Array Operations*) Let $A = \{a_1, \dots, a_n\}$ and $B = \{\text{Append, Select, Create, Copy}\}$. If $A \cap B \neq \emptyset$, a macro M is considered on $A \cup B$. The objective behind this guide is in that by separating array operations from the macro actors, the potential bottleneck of array operations will be removed, thereby speeding up execution.
5. (*Interconnection Topology*) Let $A = \{a_1, \dots, a_n\}$ such that $d_{\max} = \text{Max}\{d_{i,j}\} = 1$ for all $a_i, a_j \in A$. Let m be a dimension of hypercube interconnection network. If $m \leq n$, a set of macro $\{M_1, \dots, M_k\}$ is defined where $M_i = \{a_1, \dots, a_m\}$, $k = \lceil n/m \rceil$, and $1 \leq i \leq k$.

In what follows, we write several macro-actors to implement a simple rule. The functionality of the rule that are important to implement production systems will be taken into account. Consider a following OPS5-like rule:

```

Rule: [A (Y Z)]           ;;Condition Pattern 1
      [B (C X) (d Y)]      ;;Condition Pattern 2
      [C (p 1) (q 2) (r X)] ;;Condition Pattern 3
      →
      [Modify B (c Y) (d X)] ;;Action Pattern 1

```

Suppose that we have a Rete condition-dependency network constructed for the above rule. Fig.3 shows a conversion process of *first* condition pattern, $[A (Y Z)]$, of the rule to a macro actor. A micro data-flow graph for the comparison operations on two patterns, $[A (Y Z)]$ and $[A (B C)]$, is depicted in Fig.3(a) and the corresponding macro in Fig.3(b).

Fig.3: A conversion process for the comparison operation on two lists. (a) a micro actor data-flow graph, (b) a macro actor. WME, [A B C], is in hollow tokens and the condition pattern, [A Y Z], is in filled tokens.

The guidance rule *Guide3* is applied to this conversion process as follows: Let A be a set of five actors $\{a_1, \dots, a_5\}$ (three comparison actors and two AND actors), and L be a list of six data tokens $\{A, Y, Z, A, B, C\}$. Let r_i be the output token of an actor a_i . Applying the definition of a dependency distance to the set A , we partition A into three sets A_1 , A_2 , and A_3 , where $A_1 = \{a_1, a_2, a_3\}$, $A_2 = \{a_4\}$, and $A_3 = \{a_5\}$. We then find $d_{\max} = 2$ because

$$\text{Max}\{d_{A_1, A_2}, d_{A_1, A_3}, d_{A_2, A_3}\} = \text{Max}\{1, 2, 1\} = 2.$$

We also observe that

$$I = I_1 \cup \dots \cup I_5 = \{L, r_1, \dots, r_5\} \text{ and } O = O_1 \cup \dots \cup O_5 = \{r_1, \dots, r_5\} \quad (1)$$

From (1), we have $I_i \subseteq O$ for $1 \leq i \leq 5$, and $O_{A_3} = r_5 \in I$. Therefore, the *Guide3* is satisfied and a macro $M = \{a_1, \dots, a_5\}$ is formed. After an application of *Guide5* to the data-flow graph for the first condition pattern of the Rete network of the above rule, we obtain a graph shown in Fig.4. Note that for the sake of simplicity, five comparison micro-actors of Fig.3(a) are replaced by an actor #1 labelled *comp* in Fig.4(a).

In Fig.4(a), there are two actors related to array operations: 'append' and 'select.' The *Guide5* states that if there exists an actor a_i in the actor set A such that $a_i \in \{\text{append}, \text{select}, \dots\}$, then a macro M is considered on the set $A - a_i$. Applying the *Guide5* to the graph partitions it into three sets of micro actors A_1 , A_2 , and A_3 , where $A_1 = \{a_1, \dots, a_{12}\}$, $A_2 = \{a_{13}\}$, and $A_3 = \{a_{14}\}$. In the first step of

the conversion process, the set of actors $\{a_1, \dots, a_5\}$ however is converted to a macro M_1 . We therefore treat M_1 as a micro for the following discussion. Partitioning the graph into three subgraphs are also depicted in Fig.4(a).

The last guidance rule we apply to the data-flow graph stems from the fact that there are six true/false actors in A_1 (see Fig.4(a)). *Guide1* states that if there is a comparison actor A which immediately affects a set of true/false actors B such that $O_A \subseteq I_B$ and $d_{A,B} = 1$, we form a macro $M = \{A, B\}$.

Let A be a macro comparison actor M_1 defined in the second step and B be a set of true/false actors, $\{a_6, \dots, a_{11}\}$. We observe from the graph that

$$O_A = \{r_5\} \text{ and } I_B = \{a, b, c, d, (A, Y, Z), r_5\} \quad (2)$$

From (2), we have $O_A \subseteq I_B$ and $d_{A,B} = 1$. Therefore, the *Guide1* is satisfied and a macro $M = \{A, B\} = \{M_1, a_6, \dots, a_{11}\} = \{a_1, \dots, a_{11}\}$. The data-flow graph shown in Fig.4(a) is now completely converted into three macros $\{M_1, M_2, M_3\}$ and one micro actor a_{11} , where $M_1 = \{a_1, \dots, a_{10}\}$, $M_2 = \{a_{12}\}$, and $M_3 = \{a_{14}\}$. The three macros are shown in Fig.4(b). The conversion process we have demonstrated thus far is for the first condition pattern of the rule. However, the same argu-

Fig.4: Converting to a macro: (a) micro-actors (b) a macro. Note that in (a) three comparison actors and two AND actors which compare two lists (A Y Z) and (A B C) are replaced by an actor *comp* for the simplicity of presentation.

ment discussed above applies to other condition patterns of the rule and we will not go into further detail of the conversion process. The set of guidelines described above is by no means a complete set. It can, however, serve as a starting point for the formation of *wfms* for other AI applications.

5. Simulation and Performance Evaluation

The *Macro Data-Flow Machine* (MDFM) was adopted as the machine model of the simulator [13]. It contains 64 PEs interconnected by a 6-dim hypercube network. The target production system, which we call a "generic production system," has 15 rules, all of which are written in micro-actors based on the parallel version of the RETE algorithm [6].

A typical OPS5-like rule was shown in previous section. Each rule has on the average 5 condition patterns, 2 action patterns, and 3 two-input nodes. Each condition pattern has on the average 3 one-input nodes and at least one variable

# PEs	SD1		PD1		SD2		PD2	
	T	L	T	L	T	L	T	L
1	23619	0	8955	0	23544	0	8912	0
2	13269	9510	4589	4017	12161	9432	4840	3792
4	7239	15738	2614	6901	5873	15078	2895	6406
8	5047	22491	1701	9643	3296	21389	1545	8935
16	3519	26568	1423	11841	2101	26822	1038	11101
32	3336	33364	1314	14157	1434	31991	763	12874

Table 1: Simulation time, *T*, and network load, *L*, for a generic production system with 15 rules executed on MDFM. SD = Sequential Distribution, PD= Parallel Distribution.

# PEs	SD1/PD1		SD1/PD2		SD2/PD1		SD2/PD2	
	T	L	T	L	T	L	T	L
1	2.6	N/A	2.7	N/A	2.6	N/A	2.6	N/A
2	2.9	2.4	2.7	2.5	2.7	2.3	2.5	2.5
4	2.8	2.3	2.5	2.5	2.2	2.2	2.0	2.4
8	3.0	2.3	3.3	2.5	1.9	2.2	2.1	2.4
16	2.5	2.2	3.4	2.4	1.5	2.3	2.0	2.4
32	2.5	2.4	4.4	2.6	1.1	2.3	1.9	2.5

Table 2: Ratio of SD to PD. PD of WMEs yields a maximum of 4.4 speedup and reduces a maximum of 2.5 times the network load over SD of the original Rete algorithm. Regardless of the number of PEs used, PD provides an average of 2.5 speedup and reduces the network load on the average 2.4 times.

No. of PEs	SD1	PD1	SD2	PD2
1	1.00	1.00	1.00	1.00
2	1.78	1.95	1.94	1.84
4	3.26	3.43	4.00	3.08
8	4.68	5.26	7.14	5.76
16	6.71	6.29	11.21	8.59
32	7.08	6.82	16.42	11.68

Table 3: Speedup, $S=T_1/T_n$, of a generic production system executed on Macro data-flow simulator. Regardless of the policies used, the macro can provide a maximum of 16.4 speedup when 32 PEs are used.

in the value-part (see 3 for details). With the guidance criteria we developed, the micro actors for the rules are written in macro actors, each of which contains on the average 50 micro actors.

Tables 1-3 show simulation time, network load, and speedup. Table 1 lists simulation time units and network load for sequential and parallel distribution of WMEs with various number of PEs. Table 2 derives the ratio of sequential distribution to parallel distribution. These simulation results are depicted in Fig.5. From the simulation results, we verify two facts: *First*, our parallel network with multiple root nodes [6] gives a substantial improvement over the original sequential RETE network. The number of groups we had among condition patterns of our generic production system is 3. The simulation time of the sequential RETE network, regardless of the number of PEs used, is almost always three times that of our parallel network, as seen from Table 1 and Fig.5(c). *Second*, the data-flow principles of execution can, not only efficiently perform the symbolic computation, but also yield an impressive performance over the conventional von Neumann model of execution for production system processing. From the speedup curve of Fig.5(e), we find that the data-flow principles of execution can indeed give 17 folds of speedup when 32 PEs are used, regardless of the type of matching algorithms.

Fig.5: Simulation results on a generic production system with 15 rules. (a) Simulation time, (b) network load, (c) Ratio of SD to PD for simulation time, (d) Ratio of SD to PD for network load, and (e) Speedup. The maximum speedup we achieved by using data-driven principles of execution reached about 17 out of 32 PEs.

6. Conclusions

In this paper, a macro-actor approach for AI problems, specifically production systems, has been demonstrated as an efficient implementation tool. Characteristics of production systems from parallel processing have been discussed to

suit the macro data-flow multiprocessor environment. A simple example on comparison operations has been explained in detail from the macro perspective. Several guidelines have been given to form *wfms*. A condition pattern of a rule in PS is converted to macros. The results of a deterministic simulation with 15 rules with more than 100 condition and action patterns on the macro data-flow simulator have revealed that the macro approach is an efficient implementation for the AI production systems. Indeed, the macro approach gives 17 fold speedup out of 32 PEs used. Furthermore, our parallel matching algorithm with multiple root nodes gives an additional speedup of 3, regardless of the machine used. Assessments of the data-flow systems on productions systems have shown effective and we are currently investigating issues related to parallel firing of multiple rules for further speedup.

References

1. Arvind, R. A. Iannucci, "Two fundamental issues in multiprocessing: the dataflow solutions" MIT Laboratory for Computer Science, TM-241, Sept. 1983.
2. J. Backus, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs" *Comm. ACM* 21, 8 (Aug. 1978), pp.613-641.
3. C.L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," in *Artificial Intelligence* 19, September 1982, pp.17-37.
4. J.-L. Gaudiot and M.D. Ercegovac, "Performance evaluation of a simulated dataflow computer with low-resolution actors," *J. of Parallel and Distributed Computing*, Academic Press 1985, pp.321-351.
5. J.-L. Gaudiot and A. Sohn, "Data-Driven Multiprocessor Implementation of the Rete Match Algorithm," in *Proc. Int. Conf. on Parallel Processing*, Aug. 1988, pp. I256-260.
6. J.-L. Gaudiot and A. Sohn, "Data-Driven Parallel Production Systems," *IEEE Transactions on Software Engineering*, March 1990.
7. A. Gupta, "Parallelism in Production Systems," Morgan Kaufmann Publishers, Inc., 1987.
8. A. Gupta, C.L. Forgy, D. Kalp, A. Newell, and M. Tambe, "Parallel OPS5 on Encore Multimax," in *Proc. Int. Conf. on Parallel Processing*, Aug. 1988, pp.271-280.
9. A. Gupta and M. Tambe, "Suitability of Message Passing Computers for Implementing Production Systems," in *Proc. AAAI-88*, pp.687-692.
10. D.P. Miranker, "TREAT: A New and Efficient Match Algorithm for AI Production Systems," Morgan Kaufmann Publishers, Inc., 1989.
11. D. Moldovan, "Rubik: A Multiprocessor Rule-Based Production Systems," *IEEE Transactions on SMC* 19, July/August 1989, pp.699-706.
12. M.F.M. Tenorio and D.I. Moldovan, "Mapping Production Systems into Multiprocessors," in *Proc. Int. Conference on Parallel Processing*, August 1985, pp.56-62.
13. N. Yoo and J.-L. Gaudiot, "A Macro Data-flow Simulator," USC, Dept. of E.E.-Systems, CE:TR-89-27.
14. B.W. Wah, M.B. Lowrie, and G.-J. Li, "Computers for Symbolic Processing," *Proceedings of the IEEE* 77:4, April 1989, pp.509-540.

**DATE
FILMED**

11 / 19 / 93

END