A Macro Actor/Token Implementation of Production Systems on a Data-flow Multiprocessor+

Andrew Sohn and Jean-Luc Gaudiot
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-0781, U.S.A.
sohn@priam.usc.edu, gaudiot@priam.usc.edu

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 of the production system paradigm are identified, based on which we introduce the concept of *macro tokens* as a companion to macro actors. Aset of guidelines 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 our Macro Data-flow simulator. Simulation results demonstrate that the macro approach can be an efficient implementation of production systems.

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 it has been pointed out [Forgy, 1982], the time taken to match patterns over a set of rules can reach 90% of the total computation time spent in the processing of expert systems. This need for faster execution of production systems has spurred research in both the software and hardware domains.

From the software perspective, not only the matching step, but also the parallel firing of many productions have been studied. The Rete match algorithm has been developed to utilize the temporal redundancy in production systems [Forgy, 1982]. Further optimization of the Rete algorithm has been studied in the TREAT algorithm [Miranker, 1989], which supports the conflict set. Parallelization of the Rete algorithm has been reported to suit the multiprocessor environment [Tenorio and Moldovan, 1985].

t This work is supported in part by the U.S. Department of Energy, under Grant No. DE-FG03-87ER25043

From the hardware perspective, many studies have been reported, including shared memory multiprocessors and message passing architectures [Gupta and Tambe, 1988]. The performance of the conventional control-flow model of execution is however limited by the "von Neumann bottleneck" [Backus, 1978]. Indeed, architectures based on this model cannot easily deliver large amounts of parallelism [Arvind and Iannucci, 1983]. 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 [Gaudiot and Sohn, 1990; Gaudiot and Bic, 1991].

In this paper, we further explore the applicability of data-flow principles of execution to production systems. It has been our observation that AI problems exhibit a behavior characteristically different from conventional numeric computations. We demonstrate in this paper that a macro actor/token approach will best match these characteristics. We shall start our discussion in section 2 by introducing two fundamental approaches to AI processing. Section 3 describes those characteristics of production systems from the parallel processing perspective, which we optimize by the utilization of macro data-flow principles. A brief analysis is presented to show *why* medium grain macro actors are preferred to fine grain micro actors. Section 4 discusses several strategies about *how* to derive well-formed macro actors from micro actors for production systems. Section 5 gives simulation results based on our execution model, the macro data-flow simulator. Performance evaluation is also discussed in the section. Conclusions as well as future research issues are offered 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 infer-

ence cycle which consists of three steps: pattern matching, conflict resolution, followed by rule firing. The inference engine halts either when no rules can be satisfied 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, r, 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.I (a). The global latency, *T,* depicted in Fig.I(a), is the processing time incurred for searching the state space. Given an initial state, the inference engine finds the next state by executing an inference cycle. Based on some heuristic control strategics, the system decides which state in the search tree should be explored. The global latency *T* is thus linearly proportional to the number of states *n* to be explored in the search tree.

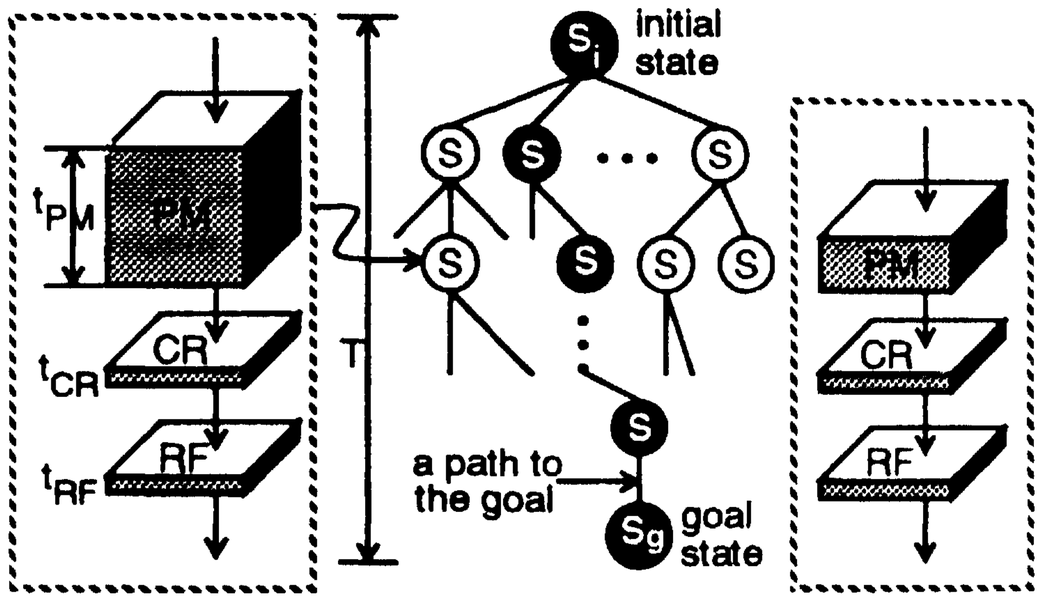


Fig.1: A search tree consisting of inference cycles, (a) an inference cycle before parallel processing, (b) after parallel processing. PM, CR, and RF stand respectively for pattern matching, conflict resolution, and rule firing

Techniques to reduce the global latency *T* in the PS paradigm can be basically classified into two categories: (1) *hardware/software parallel processing,* (2) *adaptive/ heuristic processing* [Wah *et al.,* 1989]. A straightforward technique would be to use as many 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 would hopefully find a solution in a finite amount of time. However, this technique is clearly impractical and too costly since for most AI problems the number of possible states in the search tree would be exponential even for modestly sized problems.

A way of reducing T from the adaptive/heuristic perspective is to prune unpromising branches in the search tree by deriving *heuristics* at compile time (or learning them at run time) and applying them. This second approach has been investigated by implementing neural network production systems [Sohn and Gaudiot, 1990a; 1990b] and will not be considered here since it is beyond the scope of this paper.

Our approach is centered around the data-flow principles of execution, more specifically, the *macro* data-flow principles [Gaudiot and Ercegovac, 1985]. As we shall see below, PSs exhibit distinctive characteristics. Indeed, one of the characteristics found in pattern matching is a list processing from which medium grain parallelism can be extracted.

3. Macro Data-flow Principles

A macro actor is a collection of scalar instructions. The objective behind lumping instructions into one larger unit is to improve performance by exploiting locality within these larger units. Similarly, a macro token is a collection of primitive data tokens. Consider an assertion is(x Y). This assertion, when implemented, can be represented as a list of three elements (is),(x),and Y). If we break it into three elements and form three data tokens (is), (x), and (Y) as basic elements to operate on, each of these three tokens carry little useful information.

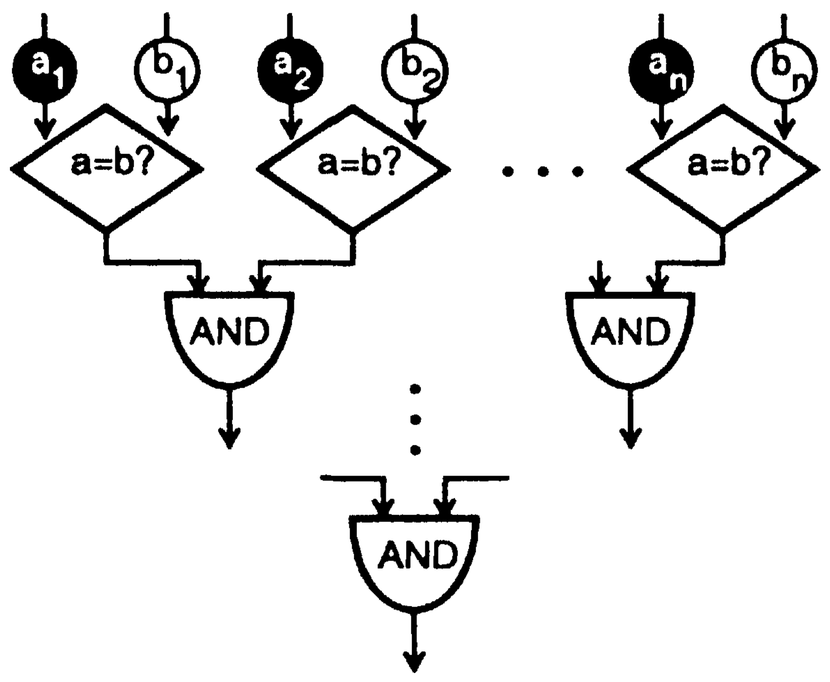


Fig.2: A data-flow graph in *micro* actors for the comparison of two lists (a_1...,a_n) and (b_1,...,b_n). There are *n/2* comparison actors to obtain a *maximum* parallelism.

When viewed from the architectural perspective, macro actors will substantially reduce the overhead in matching tags of data tokens. When using dynamic data-flow principles [Arvind and Iannucci, 1983], tokens carry tags which coasist of the context, code block, or instance of a loop to which the token belongs. 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 no less tnan three time units. However, when the two facts are compared in two lists, the tag matching time is *only* 1!

Consider a typical match operation, shown in Fig.2, which compares two lists, *(a_1,...,a_n)* and *(b_l,...,b_n).* 7o 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 a (log n)-dim hypercube. Assume that two neighboring PEs must communicate through three facilities (two communication nodes and a link) and that each PE consists of four facilities connected in a pipeline fashion. If each facility take t to execute, the total time to process *n* comparisons on *n* PEs would be

'micro, n

$= 4(1+\lceil \log_2 n \tau \rceil)\tau + 3\lceil \log_2 n \rceil \tau = (4+7\lceil \log_2 n \rceil)\tau$

where t_c=comparison time, t_a=addition time, and t_r=routing time. Note that in this simple calculation, it is assumed that no token waits in the matching/store unit of each PE. Furthermore, all the comparison actors are *ideally* allocated to neighboring PEs (which may not be realizable). The total time to compare 2 lists on 1 PE for the macro approach becomes

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

The ratio of the time taken for macro actors with 1 PE to micro actors with n PEs is

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

Note that in the micro-actor approach, we assumed that the token routing would be done in 1 step, i.e., 3r. In general, such one-step routing is impractical for a 6-dim hypercube topology. Considering that the average number of elements in a list is five for production systems, the ratio becomes R~5/(log5) -1.7. The macro actors will outperform since there is *no* communication overhead involved in macro actors (for details, see [Sohn and Gaudiot, 1991]).

There are, however, drawbacks in using macro actors. Putting too many micro actors into a macro actor will decrease the degree of parallelism, resulting in some performance degradation. Conversely, forming a macro actor with too few micro actors will not give a noticeable improvement in performance. However, the PS paradigm which we are considering for our application domain provides data parallelism which exists in many patterns and WMEs. By putting too many micro actors into a macro actor, the data parallelism will likely diminish. There must be a set of guidance criteria for the formation of macro actors. 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 macro actors.

4. Formation of Macro Actors

4.1. Guidelines for well-formed macro actors

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

An objective behind setting such a ratio is in the fact that if the processing time of the macro actor is not more than twice of the processing time for the corresponding micro actors in an ideal environment, we shall form a macro actor 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 macro actons would outperform because of no data token routing, no waiting for the mating data token (for two operand instructions), etc. In this paper, we simply set ε to 2 for macro actor formation. We now briefly describe the formation of well-formed macro actors (*wfms*).

Let I_i be a set of tokens input to a_i and O_i be a set of tokens output from 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 $\angle$ is a dependence operator which 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, $B=\{b_1,...,b_m\}$, where $b_i=\{a \mid a_i \angle a_j$ 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 five guidelines for the formation of *wfms* (the proof of correctness can be found in [Sohn and Gaudiot, 1991]):

1. (*Flow Dependency*) Let a_i and a_j be two actors. A macro actor $M=\{a_i,a_j\}$ can be defined if $O_i \subseteq I_j$ and $d_{i,j}=1$, where O_i is an output of a_i, I_i 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 $\{b_1,...,b_n\}$. A macro actor $M=\{a,b\}$ can be defined if $O_a \subseteq I_b$ and $d_{a,b}=1$. This guideline eliminates unnecessary true/false actors.
3. (*List Processing*) Let $A=\{a_1,...,a_n\}$, and L be a list of m data tokens $\{t_1,...,t_m\}$. Let $I=I_1 \cup ... \cup I_n$ and $O = O_1 \cup ... \cup O_n$. A macro actor $M=\{a_1,...,a_n\}$ can be defined if $I_i \subseteq I \cup O \cup L$ for $1 \le i \le n$ and $O_k \notin I$ for $k=d_{max}$. This guide preserves the semantics of a list.
4. (*Array Operations*) Let $A=\{a_1,...,a_n\}$ and B = { Append, Select, Create, Copy}. If $A \cap B \neq \emptyset$, a macro actor M can be formed on $A-B$. Separating array operations from the macro actors removes the potential bottleneck in array operations.
5. (*Interconnection Topology*) Let $A=\{a_1,...,a_n\}$ such that $d_{max} = \mathrm{Max}\{d_{i,j}\} = 1$ for all $a_i,a_j \in A$. Let m be a dimension of hypercube interconnection network. If $m \le n$, a set of macro actors $\{M_1,...,M_k\}$ is defined where $M_i=\{a_1,...,a_m\}$, $k=\lceil n/m \rceil$, and $1 \le i \le k$.

42. An example on the conversion process

Using these five guidelines, we shall now 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 the following OPS5-like rule:

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

Suppose that we have a Rete condition-dependency network constructed for the above rule [Gaudiot and Sohn, 1990]. Fig.3 shows a conversion process for the *first* condi-

tion element. A micro data-flow graph for the comparison operations on two elements, [A (Y Z)] and [A (B C)], is depicted in Fig.3(a) and the corresponding macro actor in Fig.3(b).

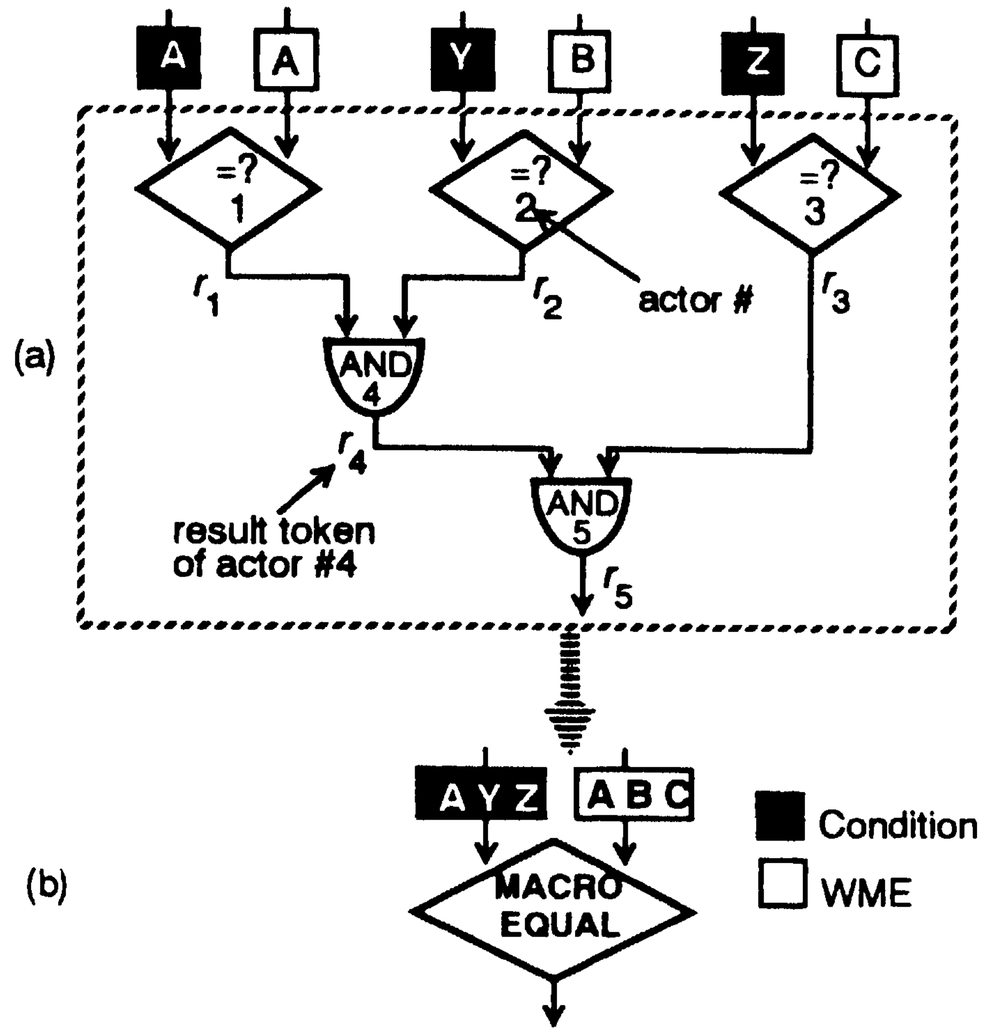


Fig.3: A conversion process. (a) a data-flow graph in micro actors, (b) a macro actor.

Rule *Guide3* is applied to this conversion process as follows: Let A be a set of five actors $\{a_1,...,a_5\}$ (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 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 $\text{Max}\{d_{A1,A2}, d_{A1,A3}, d_{A2,A3}\} = \text{Max}\{1,2,1\} = 2$. We also observe that

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

From (1), we have $I_i\subseteq I\cup O$ for $1\le i\le 5$, and $O_{A3}=r_5\notin I$. Therefore, *Guide3* is satisfied and a macro actor $M = \{a_1,...,a_5\}$ can be formed. After *Guide5* is applied to the data-flow graph for the first condition element 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) have been replaced by a single actor #1, *comp*, in Fig.4(a).

In Fig.4(a), there are 2 actors related to array operations: 'append' and 'select.' *Guide5* states that if there exists an actor a_i in A such that $a_i\in\{\text{append,select,...}\}$, then a macro actor M is considered on the set $A-a_i$. Applying the *Guide5* to the graph partitions it into 3 sets of micro actors A_1, A_2, and A_3, where $A_1=\{a_1,...,a_{12}\}$, $A_2=\{a_{13}\}$, and $A_3=\{a_{14}\}$. In the first step of the conversion process, the set of actors $\{a_1,...,a_5\}$ however is converted to a macro actor M_1. We therefore treat M_1 as a micro actor in the following discussion. Partitioning the graph into three graphs is shown in Fig.4(a).

The last 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 $d_{A,B}=1$, then we should form a macro actor $M=\{A,B\}$.

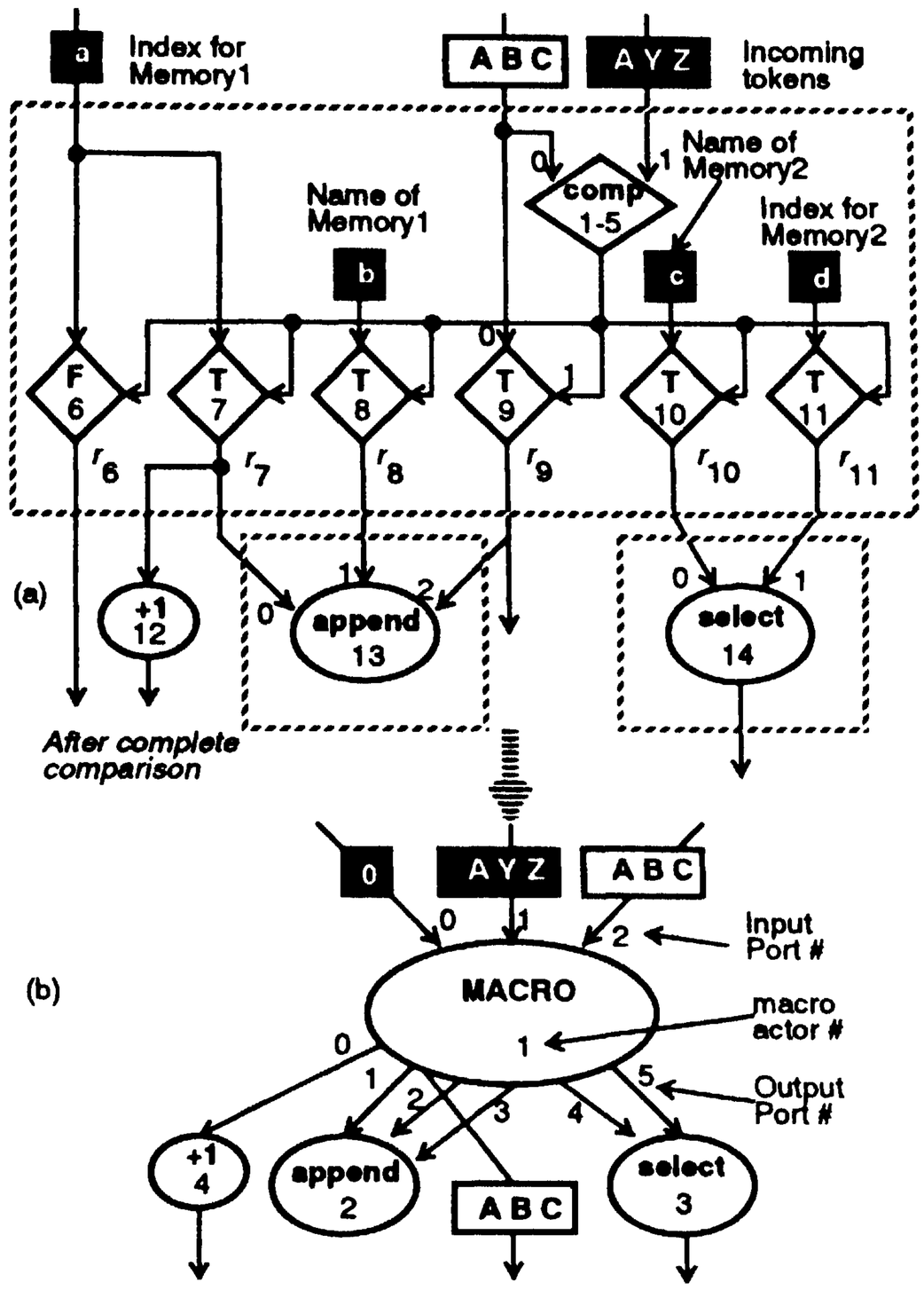


Fig.4: Converting (a) micro-actors to (b) a macro actor.

Let A be a macro comparison actor M_1 defined in the second step and B be a set of true/false actors, $\{a_6,...,a_{11}\}$. 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, *Guide1* is satisfied and a macro actor $M = \{A,B\} = \{M_1,a_6,...,a_{11}\} = \{a_1,...,a_{11}\}$ can be formed. The data-flow graph shown in Fig.4(a) is now completely converted into three macro actors $\{M_1,M_2,M_3\}$ and one micro actor a_{11}, where $M_1=\{a_1,...,a_{10}\}$, $M_2=\{a_{12}\}$, and $M_3=\{a_{12}\}$. The three macro actors are shown in Fig.4(b). The conversion process we have demonstrated thus far is for the first condition element. However, the same argument discussed above applies to other condition elements and we shall not discuss it any further. 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 applications.

5. Simulation and Performance Evaluation

A simulation has been performed on the *Macro Data-Flow Multiprocessor* (MDFM) [Yoo and Gaudiot, 1989]. The ma-

chine 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 [Gaudiot and Sohn, 1990].

A typical OPS5-like rule was shown in the previous section. Each rule has on the average 5 condition elements, 2 action elements, and 3 two-input nodes. Each condition element has on the average 3 one-input nodes and at least one variable in the value-part (see [Forgy, 1982] 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 through 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 (SD) to parallel distribution (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. Table 3 shows simulation results on speedup of using different number of PEs. Various curves for the simulations results are depicted in Fig.5.

	SD1		PD1		SD2		PD2	
# PEs	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

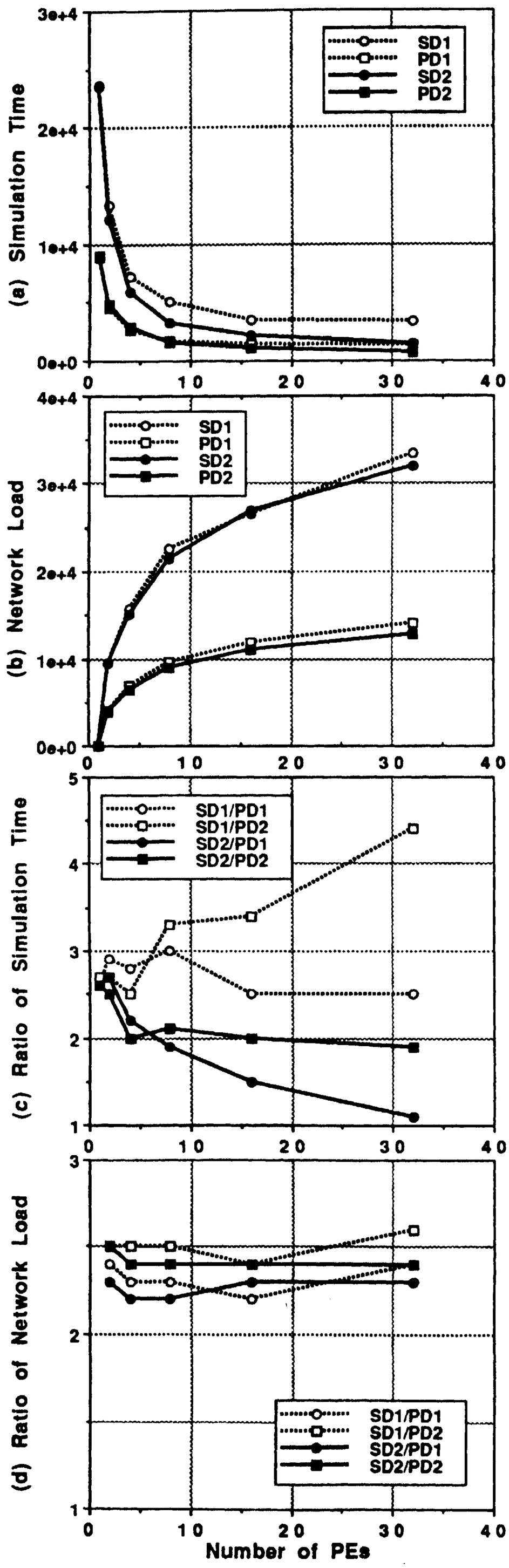
Table 1: Simulation time(*T*) and network load(*L*) for a generic production system executed on MDFM.

	SD1/PD1		SD1/PD2		SD2/PD1		SD2/PD2	
# PEs	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.

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	5.77
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.



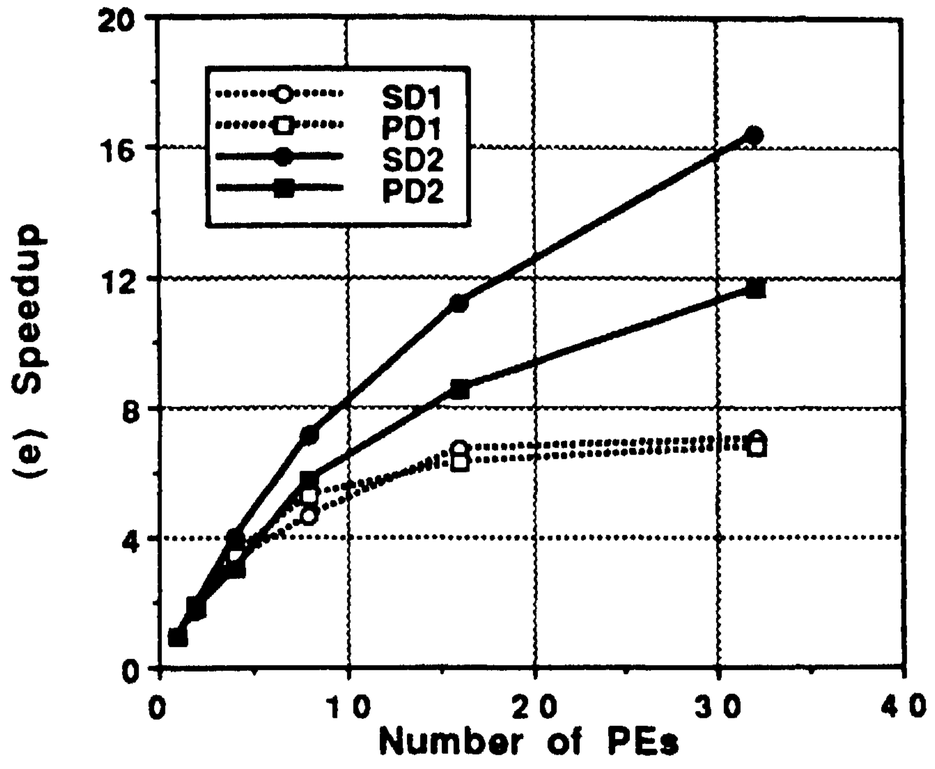


Fig.5: Simulation results on a generic production system with 15 rules, (a) Simulation time, (b) network load, (c) Ratio of SD to PD for simulation time, (d) Ratio of SD to PD for network load, and (e) Speedup. The maximum speedup we achieved by using data-driven principles of execution reached about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

From the simulation results, we verify that: *First,* our parallel network with multiple root nodes reported in [Gaudiot and Sohn, 1990] gives an impressive improvement over the original sequential RETE network: the number of groups we had among condition elements 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(c), we find that the data-flow principles of execution can indeed yield a 17-fold speedup when 32 PEs are used, regardless of the type of matching algorithms.

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 demonstrated to form *wfms.* A condition element of a rule in PS is converted to macro actors. The results of a deterministic simulation with 15 rules with more than 100 condition and action elements 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 a 17-fold speedup on 32 PEs. 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 proven effective and we are currently investigating issues related to parallel firing of multiple rules toward true parallel production systems.

References

[Arvind and Iannucci, 1983] Arvind, R. A. Iannucci, "Two fundamental issues in multiprocessing: the dataflow solutions" MIT Laboratory for Computer Science, TM-241, September 1983.

[Backus, 1978] J. Backus, "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs" C. *of ACM,* 21(8):613-641, Aug. 1978.

[Forgy, 1982] C.L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *in Artificial Intelligence,* 19:17-37, September 1982.

| Gaudiot and Bic, 1991] J.-L. Gaudiot and L. Bic, "Advanced Topics in Data-flow Computing," Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[Gaudiot and Ercegovac, 1985] J.-L. Gaudiot and M.D. Ercegovac, "Performance evaluation of a simulated dataflow computer with low-resolution actors," *J. of Parallel Distributed Computing,* pages 321-351, Academic Press 1985.

[Gaudiot and Sohn, 1990] J.-L. Gaudiot and A. Sohn, "Data-Driven Parallel Production Systems," *IEEE Transactions on Software Engineering,* 16(3)-.281-293, March 1990.

[Gupta, 1987] A. Gupta, "Parallelism in Production Systems," Morgan Kaufmann Publishers, Inc., 1987.

[Gupta and Tambe, 1988] A. Gupta and M. Tambe, "Suitability of Message Passing Computers for Implementing Production Systems," in *Proc. National Conference on AI,* pages 687-692, August 1988.

[Miranker, 1989] D.P. Miranker, "TREAT: A New and Efficient Match Algorithm for AI Production Systems," Morgan Kaufmann Publishers, Inc., 1989.

[Sohn and Gaudiot, 1990a], A. Sohn and J.-L. Gaudiot, "Connectionist Production Systems in Local Representation,"in *Proc. International Joint Conference onNeural Networks,* Washington, D.C., January 1990.

[Sohn and Gaudiot, 1990b], A. Sohn and J.-L. Gaudiot, "Representation and Processing Production Systems in Connectionist Architectures," *Int'l Journal of Pattern Recognition and Artificial Intelligence,* 4(2): 199-214, June 1990.

[Sohn and Gaudiot, 1991], A. Sohn and J.-L. Gaudiot, "Processing of Production Systems on a Macro Data-flow Multiprocessor," USC, Dept. of EE-Systems, CE:TR-91.

[Tenorio and Moldovan, 1985] M.F.M. Tenorio and D.I. Moldovan, "Mapping Production Systems into Multiprocessors," in *Proc. International Conference on Parallel Processing,* pages 56-62, August 1985.

[Yoo and Gaudiot, 1989] N. Yoo and J.-L. Gaudiot, "A Mao no Data-flow Simulator," USC, Dept. of E.E.-Systems, CE:TR-89-27.

[Wah *et al.,* 1989] B.W. Wah, M.B. Lowrie, and G.-J. Li, "Computers for Symbolic Processing," *Proceedings of the IEEE,* 77(4):509-540, April 1989..