An Associative Memory Architecture for Concurrent Production Systems

José Nelson Amaral* and Joydeep Ghosh
(*nelson@pine.ece.utexas.edu*, *ghosh@pine.ece.utexas.edu*)
Department of Electrical and Computer Engineering,
University of Texas at Austin,
Austin, Texas 78712

Abstract

This paper presents a novel parallel architecture for production systems. It relies on modern associative memory techniques to construct an environment in which productions can be fired in parallel even before the actions of previously fired productions are fully processed. This approach is made possible by the use of the serializability criterion for correctness. An approximate solution for the rule partitioning problem is presented. Measurements of speedup obtained from a detailed event-driven simulator indicate the potential of this new architecture.

1. Motivation

The interest in new research towards acceleration of knowledge systems is sustained by an extensive and profitable market that is in need of handling massive databases at ever increasing speeds [15, 6]. Kuo and Moldovan [9] and Amaral and Ghosh [2] survey various approaches to improve Production Systems (PS) execution speed.

In spite of the increasing popularity of parallel architectures, the architectures providing best cost/performance are compiled Production Systems (PS) running on general purpose uniprocessors [11]. The search for an effective parallel PS machine follows two schools of thought. The first believes that little improvement can be obtained in the performance of parallel PS without the introduction of significant changes to PS language semantics [14]. The second tries to maintain the most attractive aspect of production systems, that is, the simplicity of "OPS5-like" semantics [3], while seeking better performance through architectural improvements [10, 13].

The original OPS5 implements the commutativity criterion of correctness, which demands that a parallel implementation produces the same results as *any* sequential execution of productions [7]. The selection strategy of OPS5 allows only the most recent and specific production to fire at any cycle. Programmers often rely on this strategy to ensure correctness. The architecture presented in this paper maintains most of the OPS5 semantics; it modifies, however, the selection strategy and uses serializability as a correctness criterion [12, 13]. The serializability criterion allows any matched production to fire at any time as long as *at least one* sequential execution of the productions produces the same results as the parallel one, Serializability is less restrictive and allows more parallelism in the execution of a production system. The drawback is that

*Sponsored in part by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) - Brazil.

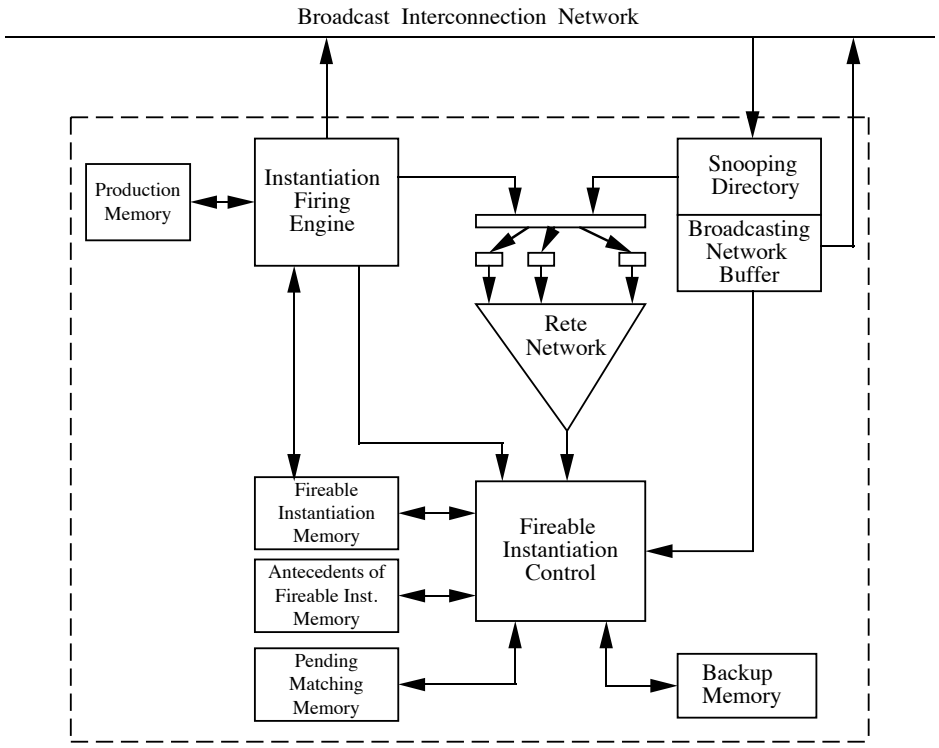

Figure 1: Processing Element Model

the programmer has to ensure that, at any given time, every possible execution sequence of the enabled productions generates correct results.

2. Parallel Architecture

We propose a novel parallel architecture for compiled PS that allows concurrent firing of productions. This architecture is formed by a number of identical processors connected through a Broadcast Interconnection Network (BIN)[1]. Each of the processors has the internal organization shown in Figure 1. An I/O processor attached to the BIN initially loads the productions and the initial database in the processors. At compile time each production is uniquely assigned to a processor according to a partitioning algorithm that takes into consideration inter-production dependencies and workload balance. Each processor stores locally all Working Memory Elements (WME) that are tested by its production antecedents.

All tokens propagated over the BIN consist of deletion, addition or modification of a WME. Such operations might enable or disable a local production. Upon processing a token, the Fireable Instantiation Control (FIC) has to do the following: perform an associative search in the Antecedents of Fireable Instantiation Memory (AFIM) to verify which previously enabled productions are now disabled; remove such productions from the Fireable Instantiation Memory (FIM), and remove all their antecedents from AFIM; place the incoming token in the input queue of the Rete Network. Notice that because the productions that are no longer fireable were removed from FIM, a *partially informed* selection can proceed and select a new production to be

[1]This network might be implemented as a bus.

fired among the ones that remained in FIM.

This capability to fire a new production before the changes generated in the previous production firing are fully propagated through the Rete Network results in low overhead for token removing[2], and allows the maintenance of the beta memories of the original Rete algorithm [4]. This combination of the advantages of Rete and Treat is made possible by the storage of negated conditions in the representation of fireable instantiations of productions stored in FIM.

The compiler classifies the productions as local or remote: a local production modifies only WMEs that are exclusively stored in local memories; a remote production changes pieces of memory that are stored in other processors. To select a production to fire, the Instantiation Firing Engine (IFE) performs an associative search in FIM to find the most recently enabled production. If the selected production is remote, the IFE places a request for ownership of the BIN. Upon receiving BIN ownership, the IFE waits until all outstanding tokens from previous broadcastings are processed by FIC. The IFE access FIM to verify whether the selected production is still fireable. If it is, IFE proceeds to execute its actions, propagating tokens that change shared WMEs in BIN and sending tokens that modify only local WMEs to FIC and Rete.

The Snooping Directory (SD) is an associative memory that contains a list of all WME types that are tested by antecedents of the productions assigned to the local processor. SD "snoops" BIN and capture only tokens that modify WMEs relevant to the processor. If there is a local production being executed, the token cannot be immediately processed. It is stored in the Broadcasting Network Buffer (BNB), and is processed as soon as the local production processing finishes.

The Pending Matching Memory (PMM) is necessary to store tokens that are in the Rete Network. Whenever a change to the conflict set[3] is generated in the Rete Network, FIC performs an associative search in PMM to verify if a later modification invalidates such change. This mechanism prevents races between IFE and Rete.

[2]Low overhead in token removing is the most salient advantage of the Treat algorithm [10].

3. Partition of Rules

The partitioning of a production set into a number of processors can be modeled as the minimum cut problem, which has been proven to be NP-Complete [5]. The polynomial time approximate solution presented in this section has three goals: minimizing the duplication of working memory elements; reducing traffic in the bus; and balancing the amount of processing in each processor.

We are concerned with two kinds of relationships among productions: productions that share antecedents, and productions that have conflicting actions[4]. Assigning productions with common antecedents to the same processor reduces memory duplication, while assigning productions with conflicting actions to the same processor prevents traffic in the bus.

To account for these two kinds of relationships we define an undirected, fully connected graph PRG $= (P, E)$ called *Production Relationship Graph*. Each vertex in P represents one of the productions in the system, and each edge in E is a combined measure of the production relationships. PRG has a weight function $w : E \rightarrow Z^+$, defined by equation 1.

$$w(E_{ij}) = w(E_{ji}) = (1-\delta_{ij}) \sum_{l=0}^{n-1} \sum_{k=0}^{m-1} \psi_{li,kj} + (1-\delta_{ij}) \sum_{l=0}^{p-1} \sum_{k=0}^{q-1} \gamma_{li,kj}, \quad (1)$$

where n is the number of antecedents in production R_i, m is the number of antecedents in

[3]"Conflict set" is the set of all productions enabled to be fired at any given time.

[4]Two productions have a conflicting action if one creates and the other destroys the same WME.

production R_j, p is the number of consequents in production R_i and q is the number of consequents in production R_j, δ_{ij} is 1 if $i = j$ and 0 otherwise, and

$$\psi_{li,kj} = \begin{cases} 1 & \text{if antecedents } A_l \text{ of } R_i \text{ and} \\ & A_k \text{ of } R_j \text{ are of the same type.} \\ 0 & \text{otherwise} \end{cases}$$

$$\gamma_{li,kj} = \begin{cases} 1 & \text{if consequent } W_l \text{ of } R_i \\ & \text{conflicts with } W_k \text{ of } R_j \\ 0 & \text{otherwise} \end{cases}$$

Simulation results show that the main factor limiting further speedup is the time spent in the matching phase in the Rete network. Consequently, the load balancing must concentrate in the processing performed in the Rete network. Furthermore, most of the time in the Rete network is spent in β-node activities. Thus, the number of β-tests performed in the antecedents of a production is used as a measure of the work load associated with this production. To address the constraint of balancing the amount of processing among processors, we define the function $B : P_0, ..., P_{N-1} \rightarrow Z^+$, which computes the number of beta tests that are expected to be performed by processor P_i.

$$B(P_i) = \sum_{j=0}^{k} \beta(R_j)\, \varphi_{ij}, \qquad (2)$$

where k is the number of productions assigned to processor P_i, $\beta(R_j)$ is the number of beta tests performed for production R_j, and φ_{ij} is 1 if R_j is assigned to P_i, and 0 otherwise[5].

The strategy used in this partitioning algorithm consists of selecting the processor with the least number of estimated beta tests, and then choosing the production best fitted to this processor. The fitness of a given production R_i to a processor P_k is measured by the value of the function $F(R_i, P_k)$.

$$F(R_i, P_k) = \sum_{j=0}^{N-1} w(E_{ij})\, \eta_{jk}, \qquad (3)$$

$$\eta_{jk} = \begin{cases} 2 & \text{if } R_j \in P_k \\ 1 & \text{if } R_j \in P \\ -1 & \text{if } R_j \in P_m \neq P_k \end{cases}$$

With the functions $B(P_i)$ and $F(R_i, P_k)$ defined, we can present the algorithm. In the beginning all productions R_k are in the set P, and all subsets P_i are empty. The productions strongly related to other productions in PRG are the first ones to be assigned to processors. At each pass, the processor with the least load is assigned the production best fitted to it.

PARTITION(P, E, w, N, B, F)
1 **while** $P \neq \emptyset$
2 **do** $P_k \leftarrow P_k \cup \{\ R_i\ /R_i \in P$ and
$B(P_k) = \min_k B(P_k)$ and
$F(R_i, P_k) = \max_i F(R_i, P_k)\}$
3 $P \leftarrow P - \{P_i\}$

4. Performance Evaluation

To evaluate the performance of the architecture presented in this paper, we have developed a detailed event driven simulator. The unavailability of a representative set of benchmarks is a well known weakness in the evaluation of performance of novel production systems. An added difficulty with this architecture is the use of the serializability criterion. The few OPS5 benchmarks available in the research community rely on the selection strategy to guarantee correctness. We have developed a new benchmark that is a modified version of the well known Traveling Salesperson Problem (TSP). In our version, the

[5] $\beta(R_j)$ is an estimate of the number of beta tests performed because of the presence of production R_j. It is measured in previous running of the same production system.

cities are grouped by country, and the salesperson can enter each country only once. By varying the number of countries and the number of cities per country, the researcher can vary the amount of local and shared data[6].

Bench.	# Prod	Ant./prod	Cons./prod
`life`	41	6.1	1.3
`hotel`	80	4.1	2.0
`patents`	86	5.2	1.2
`south2`	121	4.7	2.7
`moun2`	301	4.7	2.7

Table 1: Benchmarks and Speedup.

Table 1 shows static measures — number of productions, average number of antecedents per production, average number of consequents per productions — for the benchmarks used. `south2` has four countries and ten cities per country; `moun2` has ten countries and 15 cities per country. `life` is the game of life implemented by Anurag Acharya at CMU. `hotel` is a production systems that simulates the management and running of a hotel. It was implemented by Steve Kuo at USC, and modified by Anurag Acharya and by José Amaral. `patents` is our solution to the "confusion of patents problem" [8].

The speedup curves in Figure 2 show that the speedup varies with the benchmark. Certain applications have more intrinsic parallelism than others. For example, the modified TSP problem with ten countries (`moun2`) produces significantly more speedup than the one with four countries (`south2`). It is remarkable that for most of the benchmarks, the efficiency drops significantly when more than ten processors are used, indicating that if resources are to be used efficiently, we should use about ten processors in this architecture.

We are currently evaluating the use of multiple matching units within the Rete network to reduce the amount of time spent in matching. Our preliminary results indicate that such improvement could obtain a four-fold speedup. This speedup is a multiplicative factor to the speedup shown in Figure 2.

[6]This benchmark is available to the research community. We intend to present it in greater detail in a future publication.

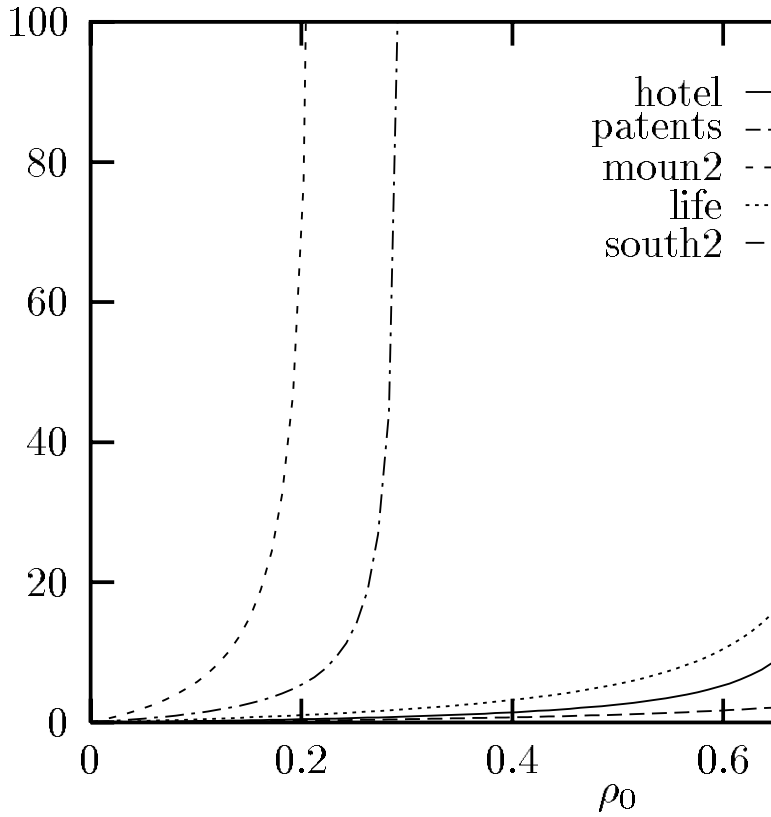

Figure 2: Speedup Curves

5. Conclusion

This architecture combines the gains of recent developments in compiling techniques, dependency analysis, associative memories, snooping caches and bus arbitration strategies. It also can be used as a basic building block for a larger, hierarchical, and possibly scalable architecture. We have developed a comprehensive software environment along with a detailed event driven simulator for the architecture outlined above. Several comparative performance results were

presented showing the potentials and limitations of this novel approach to improving PS speed.

6. Acknowledgements

We are very thankful to Anurag Acharya for letting us use the front-end of his parallel compiler [1], for being so helpful with many questions, and for providing some of the benchmarks that we used. We also would like to acknowledge the help of Howard Owens on tracking a difficult bug in the implementation of the simulator, and thank Dan Miranker for fruitful discussions.

References

[1] A. Acharya, M. Tambe, and A. Gupta. Implementation of production systems on message-passing computers. In *IEEE Trans. on Parallel and Distributed Systems*, volume 3, pages 477–487, July 1992.

[2] J. N. Amaral and J. Ghosh. Speeding up production systems: From concurrent matching to parallel rule firing. In L. N. Kanal, V. Kumar, H. Kitani, and C. Suttner, editors, *Parallel Processing for AI*. Elsevier Science Publishers B.V., 1994.

[3] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*. Addison-Wesley, Massachusetts, 1985.

[4] C. L. Forgy. *On the Efficient Implementations of Production Systems*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1979.

[5] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comput. Sci.*, 1:237–267, 1976.

[6] F. Hayes-Roth and N. Jacobstein. The state of knowledge-based systems. *Communications of the ACM*, 37(3):26–39, March 1994.

[7] T. Ishida and S. Stolfo. Towards the parallel execution of rules in production system programs. In *Proceedings of International Conference on Parallel Processing*, pages 568–575, 1985.

[8] P. C. Jackson Jr. *Introduction to Artificial Intelligence*. Dover Pub., New York, 1985.

[9] S. Kuo and D. Moldovan. The state of the art in parallel production systems. *Journal of Parallel and Distributed Computing*, 15:1–26, June 1992.

[10] D. P. Miranker. *TREAT: A New and Efficient Match Algorithm for AI Production Systems*. Pittman/Morgan-Kaufman, 1990.

[11] D.P. Miranker and B. J. Lofaso. The Organization and Performance of a TREAT Based Production System Compiler. *IEEE Trans. on Knowledge and Data Engineering*, pages 3–10, March 1991.

[12] D. E. Neiman. Control issues in parallel rule-firing production systems. In *Proceedings of National Conference on Artificial Intelligence*, pages 310–316, July 1991.

[13] J. G. Schmolze. Guaranteeing serializable results in synchronous parallel production systems. *Journal of Parallel and Distributed Computing*, 13:348–365, December 1991.

[14] S. Stolfo, H. Dewan, and O. Wolfson. The PARULEL parallel rule language. In *Proc. 1991 International Conference on Parallel Processing*, pages 36–45, 1991.

[15] A. C. Stylianou, G. R. Madey, and R. D. Smith. Selection criteria for expert system shells: A socio-tecnical framework. *Communications of the ACM*, 35(10):30–48, October 1992.