Associative Memories Provide an Efficient Control Mechanism for a Parallel Production System Architecture

José Nelson Amaral*
(*amaral@madona.pucrs.br*)
Departamento de Eletrônica
Pontifícia Universidade Católica do RGS
90619-900 - Porto Alegre, RS

Joydeep Ghosh†
(*ghosh@pine.ece.utexas.edu*)
Dept. of Electr. and Comp. Engineering
The University of Texas at Austin
Austin, Texas 78712

Abstract

Recently we proposed a parallel architecture for production systems [2, 3, 5]. This novel architecture allows parallel production firing, concurrent matching, and overlap among matching, selection, and firing of productions. The elimination of global synchronization in production systems was made possible by the use of serializability as a correctness criterion and by the construction of an efficient control mechanism for the operation of the machine. This control mechanism relies heavily in the use of associative memory devices as lookaside tables. In this paper we study the impact of these memories in the overall performance of the architecture. We also estimate the amount of associative memory needed for a typical production system. These results, obtained from a comprehensive system level event-driven simulator, indicate that substantial improvements in speed can be achieved with a very modest increase in hardware cost.

1 Introduction

Considerable efforts have been made towards speeding up production system machines in the past twenty years [4, 10]. Originally, production systems were realized as interpreted language programs for sequential machines. The high cost of matching motivated the development of concurrent matching systems and, subsequently, systems that also allowed multiple productions to be fired at the same time [12]. In a separate line of research, modern compile optimization techniques were developed to run production system programs more efficiently on general purpose sequential machines [9].

These efforts have led to great advances in the understanding of the issues involved in the construction of faster production system machines, but only limited improvement in actual

*Supported by a fellowship from Conselho Nacional de Desenvolvimento Cient´ıfico e Tecnológico (CNPq) and by Pontif´ıcia Universidade Católica do Rio Grande do Sul (PUCRS) - Brazil.

†Supported in part by NSF under grant ECS-9307632 and by Faculty Development Awards from TRW Foundation and Schlumberger.

performance. Also, there have been few attempts to integrate progress made in different areas: (1) the use of the restrictive commutativity criterion for correctness and the notion of a match-select-act "cycle" forced even advanced architectures to perform synchronization before each production firing; (2) compile optimization techniques were usually restricted to sequential machines; (3) many of the concurrent matching engines were constructed with a large number of small processors and were not combined with parallel firing techniques. Moreover, parallel processing researchers failed to take advantage of the fact that, in typical production systems, reading operations are performed much more often than writing ones.

We propose a novel parallel production system architecture that uses the less restrictive serializability criterion for correctness. This architecture eliminates the concept of a production system "cycle", thus eliminating the need to construct a global "conflict set" and to perform global synchronization before each production firing. Productions are partitioned among processors based on information about the workload of each production and on production dependencies identified through compiling techniques. A production can be selected to fire before all the matches resulting from previous production actions are complete. This papers discusses the use of associative memories as supporting devices for the control mechanism of the architecture. A key structure in the design of this new architecture, content addressable memories allows for a quick verification of which actions can be performed in parallel.

2 Parallel Architecture

The proposed 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

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

were removed from FIM, a *partially informed* selection can proceed and select a new production to be fired among the ones that remained in FIM.

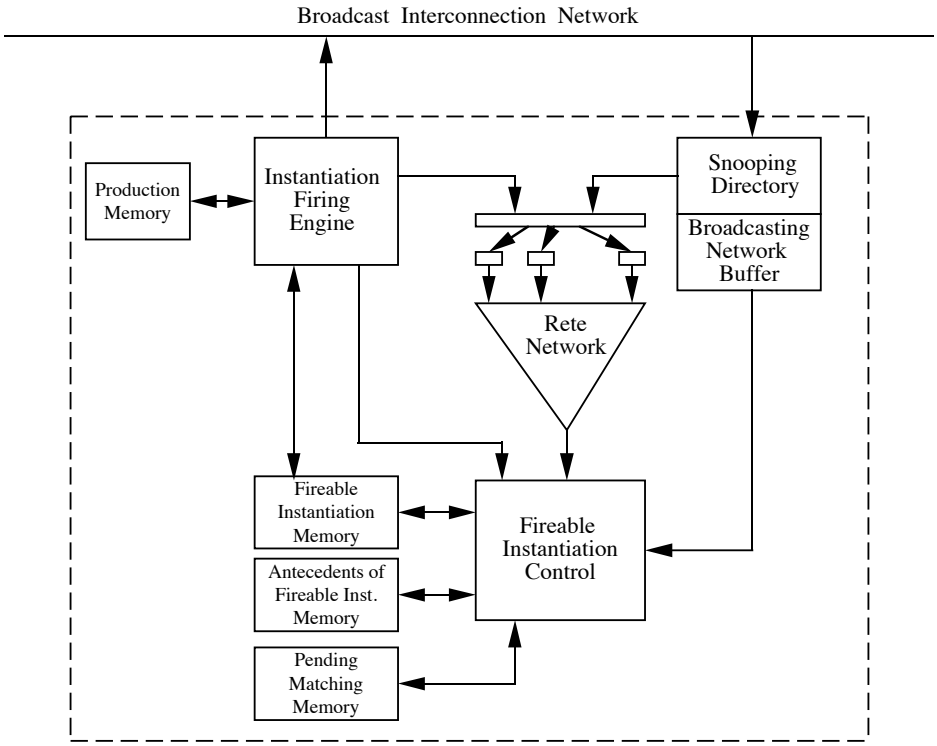

Figure 1: Processing Element Model

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 [7]. 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

[2]Low overhead in token removing is the most salient advantage of the Treat algorithm [11].

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.

A key feature of this architecture is the use of modern associative memory technology. For example, the Snooping Directory is an associative memory used by each processor to verify whether broadcast changes need to be captured for local processing. The use of the associative Antecedents of Fireable Instantiation Memory (AFIM) allows the quick elimination from the Fireable Instantiation Memory (FIM) of instantiations that are no longer fireable. Consequently the Instantiation Firing Engine (IFE) can select and fire an instantiation before the actions of the preceding production are fully processed through the Rete Network. Another piece of associative Memory, the Pending Matching Memory (PMM) is necessary to prevent racing between the IFE and the Rete Network. Section 3 presents a brief report on the performance evaluation of the architecture[4]. We dedicate the rest of the paper to the study of the impact of the use of associative memories in the performance of the architecture.

3 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 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 [2].

Table 1 shows static measures — number of productions, number of distinct WME types, average number of antecedents per production, average number of consequents per productions

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

[4] An extensive description of the architecture and of is performance measurements can be found in [2, 4, 5].

— for the benchmarks used to estimate performance in the multiple functional unit Rete network. `south` and `south2` are CTSPs with four countries and ten cities per country; `moun` and `moun2` are CTSPs with ten countries and 15 cities per country.

Bench.	# Prod	Ant./prod	Cons./prod	# WME types
`life`	40	6.1	1.3	5
`hotel`	80	4.1	2.0	62
`patents`	86	5.2	1.2	4
`south`	91	4.7	2.8	40
`south2`	121	4.7	2.7	61
`moun`	211	4.7	2.8	88
`moun2`	301	4.7	2.7	151
`waltz2`	10	2.7	8.0	7

Table 1: Static Measures for Benchmarks Used.

`patents` is our solution to the constraint *Confusion of Patents Problem* presented in [6]. Because this solution has only four different types of WMEs, most of the productions either change or test the same kinds of WME. As a consequence, productions have strong interdependency, resulting in a production system poorly suited for clustering. The main source of parallelism is the concurrent execution of different portions of the Rete network. Originally written by Steve Kuo at the University of Southern California, `hotel` is a production system that models the operation of a hotel. It is a relatively large and varied production system (80 productions, 65 WME types) with 17 non-exclusive contexts. `life` is an implementation for Conway's game of life, as constructed by Anurag Acharya. After our modifications, `life` has forty productions. Twenty five of these productions are in the context that computes the value of each cell for the next generation and potentially can be fired in parallel. The other fifteen productions are used for sequencing and printing and can be only slightly accelerated by Rete network parallelism. Our version of the line labeling problem, `waltz2`, was originally written by Toru Ishida (Columbia Univ.), and successively modified by Dan Neiman (Univ. of Massachusetts), Anurag Acharya (Carnegie-Mellon Univ.) and José Amaral (Univ. of Texas). It has two non-overlapping stages of execution, each one with four productions.

The benchmarks described in table 1 were used to evaluate the performance of the proposed architecture. First we measured the amount of speedup over an architecture with global synchronization and without overlapping between matching and selecting-acting within a processor. Then we investigate the effectiveness of the use of associative memories. Finally we obtain estimates for the size of associative memories needed for each one of the benchmarks and for the level of activity in the bus.

3.1 Parallel Firing Speedup

To measure the advantages of parallel production firing and of the internal parallelism within each processor, we define a globally synchronized architecture that is very similar to the one proposed in this paper, except that it performs global conflict set resolution to implement the OPS5 recency strategy. This synchronized architecture is also very similar to the one suggested by Gupta, Forgy, and Newell [8]. In this architecture, each processor reports the best local instantiation to be fired to the bus controller. The bus controller selects the instantiation whose time tag indicates it to be the latest one to become fireable. This added decision capability in the bus controller implements the recency strategy to solve the conflict set. The processor selected to fire a production broadcasts all changes in the bus. A processor only selects a new candidate to fire when the matching in the Rete network is complete. The bus controller waits until all processors report a new candidate to fire. This mechanism reproduces the global synchronization and conflict set generation/resolution present in many of the previously proposed architectures. In order to have a fair comparison, we considered that the synchronized architecture uses an associative memory to store and solve the local conflict sets, and that the bus controller chooses the "winner" in one time step.

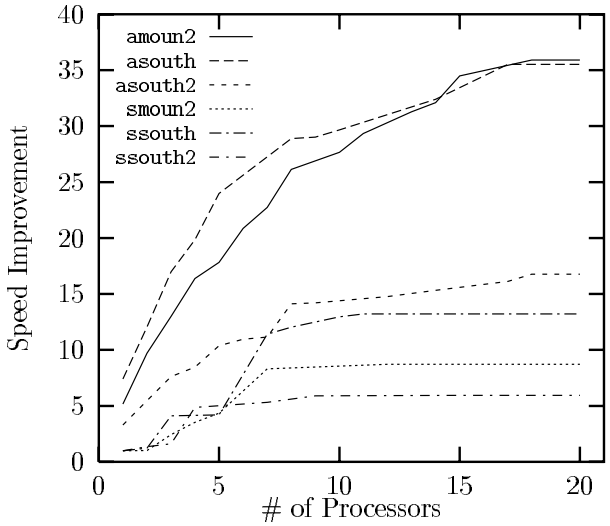

Figure 2: Speedup Curves

Figure 2 shows the comparative performance for the CTSP benchmarks[5]. Since the synchronized architecture also uses associative memory to store and search the local conflict sets, the comparisons of Figure 2 do not reflect the advantages of using such memories in our architecture. Significant speedup is observed over the synchronized architecture even for the single

[5]Due to limitation in space in this article, we will not present speed measures for the other benchmarks, see [2].

processor configuration. This measures the amount of speed that is gained due to the parallelism between matching and selecting/firing. The apparent superlinear speedup in the curves of Figure 2 reflects the fact that these curves are showing the combined speedup due to two different factors: intra and interprocessor parallelism. To obtain the speedup due exclusively to parallel production firing, the reader should divide the values in the "a" curves by the values in the same curve for a single processor machine. These results confirm our initial conjecture that the elimination of the global synchronization in a production system allows the construction of machines with significant speedup.

3.2 Effectiveness of Associative Memories

In section 2 we stated that the design of the architecture is based on the premise that the use of Content Addressable Memories (CAMs) significantly improves the processing speed. To further investigate this premise, we implemented options in the simulator that allow us to specify whether each one of the individual memory components — AFIM, FIM, and PMM — is a CAM or a traditional *Reference* Addressable Memory (RAM) [13]. The effectiveness of a CAM in the architecture depends on the amount of data stored in the memory, the frequency of access, and whether its accesses are in the critical path of execution. Thus, the amount of speedup obtained by a given combination of CAM/RAM memories depends on the production system program that the machine is executing.

To set up experiments to measure these speedups, we defined two quantities: $S_p(M, B)$ and $S_l(M, B)$. $S_p(M, B)$ is the amount of speedup (of the overall system) that results when the memory component M is replaced for a CAM in a machine that was originally formed only by RAMs. M designates one of the memory components — PMM, AFIM, or FIM — and B is a benchmark program. Similarly, when the reference machine uses only CAMs, $S_l(M, B)$ measures the reduction in speed (Slowdown) that would occur if the memory component M were to be replaced by a RAM.

Table 2 presents the average speedup for machines with one up to twenty processors. In practical designs, CAMs might be slower than RAMs for the same technology and silicon area, because of the extra logic required. Thus we introduce a *technology factor* T that indicates how much slower a basic operation such as the reading or writing of a single data element was considered in this comparison. Table 2 shows measures for a machine with CAMs with the same speed as the RAMs ($T = 1$) and for a machine with CAMs that are four times slower ($T = 4$) than RAMs. Observe that there is no significant difference in speedup between the two measures, indicating the advantage of the use of CAMs, even if they are slower than RAMs.

[6] Each number is an average of 20 values, obtained for systems with 1 through 20 processors.

Bench	T	PMM		FIM		AFIM		All
		S_p	S_l	S_p	S_l	S_p	S_l	S_p
`hotel`	1	3.0	29.3	1.0	1.6	1.6	13.5	45.5
`hotel`	4	3.0	30.1	1.0	1.6	1.5	13.6	45.3
`life`	1	2.8	2.1	1.3	1.0	1.6	1.2	3.4
`life`	4	2.8	2.1	1.3	1.0	1.6	1.2	3.4
`moun2`	1	3.2	4.9	1.0	1.0	1.8	2.5	8.5
`moun2`	4	3.3	4.9	1.0	1.0	1.7	2.5	8.5
`patents`	1	1.9	1.6	1.0	1.0	1.4	1.2	2.3
`patents`	4	1.9	1.6	1.0	1.0	1.4	1.2	2.3
`south2`	1	3.4	10.0	1.0	1.1	1.4	4.3	14.9
`south2`	4	3.3	10.2	1.0	1.1	1.5	4.4	14.8
`waltz2`	1	1.8	1.4	1.0	1.0	1.9	1.6	3.0
`waltz2`	4	1.8	1.5	1.0	1.0	1.9	1.6	3.0

Table 2: Speedup due to use of CAMs[6].

The last column of Table 2 shows the speedup that compares a configuration with all three memories associative against one in which all three memories are RAM. Table 2 shows that replacement of just one memory for a CAM results in quite low speedup, but when all three memories are made CAMs, the processing speed shows considerable improvement. Overall, these results confirm our initial conjecture that the use of CAMs can provide considerable speedup in production system architectures.

3.3 Associative Memory Size

The next question that the inquisitive computer architect must ask is: how large do these associative memories need to be? The simulator has an option to report the "crest"[7] of each memory component in any given run. Table 3 shows the maximum and the average crest over machines with up to twenty processors. The average crest is the average of the largest memory needed for each machine configuration. The maximum crest indicates the minimum memory size needed to run that specific benchmark. Observe that for some memory/benchmark the average crest is several times smaller than the maximum crest (see AFIM in `moun2` and PMM in `waltz2`). If memory size becames a concern in the construction of the machine, a RAM can be used to contain overflow. The absence of a direct correlation between the size of the memory crest in table 3 and the speedup and slowdown shown in table 2 reflects the fact that the processing speed is not solely dependent on the amount of data stored in each memory: it also depends on the frequency and time of access of these memories.

[7]The *crest* of a memory component is the maximum amount of data stored in that memory component in any processor of the machine for a given benchmark and a specified number of processors.

Benchmark	PMM		FIM		AFIM		FIM(synchronized)	
	Max	Ave	Max	Ave	Max	Ave	Max	Ave
`hotel`	3200	1436	395	216	1030	699	3580	1178
`life`	2877	2643	690	584	3313	1472	23030	8787
`moun2`	27899	23303	2580	727	15634	3042	313400	46747
`patents`	776	739	605	179	1549	449	1410	426
`south2`	4788	2822	350	95	1159	611	47205	8414
`waltz2`	3573	1109	1250	870	2797	1688	5785	3299

Table 3: Maximum and average "crest" for memory size (bytes).

The speed comparison with the synchronized architecture presented in section 3.1 considered that both architectures used associative memory to store and search the conflict set. The average and the maximum crests of the associative memories for the synchronized architecture are presented in the rightmost columns of Table 3. Observe that for most of the significant benchmarks, the synchronized architecture needs a much larger memory. For the CSTPs benchmarks (`moun2` and `south2`) the maximum crest in the synchronized architecture was ten times larger than in the architecture proposed in this paper. This evidences that the "eager firing" mechanism also reduces the demand for memory.

4 Concluding Remarks

We proposed a new architecture for production systems that eliminates global synchronization and the generation of a global conflict set. The increased importance of associative search for maintaining fireable instantiation tables in this setting is evidenced by the big performance gains obtained by using modest amounts of associative memory. Note that a single physical CAM can be logically partitioned into PMM, FIM and AFIM, and the "crests" in each partition are not expected to occur in the same processor and at the same time. Thus, only a few kilobytes of associative memory is sufficient for most of the benchmarks considered.

Acknowledgements.

We are thankful to Anurag Acharya for letting us use the front-end of his parallel compiler [1] and for providing some of the benchmarks that we used, and to 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. *A Parallel Architecture for Serializable Production Systems.* PhD thesis, The University of Texas at Austin, Austin, TX, December 1994. Electrical and Computer Engineering.

[3] J. N. Amaral and J. Ghosh. An associative memory architecture for concurrent production systems. In *Proc. 1994 IEEE International Conference on Systems, Man and Cybernetics*, pages 2219–2224, San Antonio, TX, October 1994.

[4] 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*, chapter 7, pages 139–160. Elsevier Science Publishers B.V., 1994.

[5] J. N. Amaral and J. Ghosh. Performance measurements of a concurrent production system architecture without global synchronization. In *Proc. 9th International Parallel Processing Symposium*, pages 790–797, Santa Barbara, CA, April 1995.

[6] R. E. Fikes. REF-ARF: A system for solving problems stated as procedures. *Artificial Intelligence*, 1(1):27–120, 1970.

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

[8] A. Gupta, C. Forgy, and A. Newell. High-speed implementations of rule-based systems. *ACM Transactions on Computer Systems*, 7:119–146, May 1989.

[9] C.-M. Kuo, D. P. Miranker, and J. C. Browne. On the performance of the CREL system. *Journal of Parallel and Distributed Computing*, 13:424–441, December 1991.

[10] 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.

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

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

[13] J. P. Wade. *An integrated content addressable memory system.* PhD thesis, Massachusetts Institute of Technology, May 1988.