Design and Analysis of a "Superscalar" Production System Machine

José Nelson Amaral[1]
(*amaral@madona.pucrs.br*)
Departamento de Eletrônica
Pontifícia Universidade Católica do RGS
90619-900 - Porto Alegre, RS, Brazil

Joydeep Ghosh[2]
(*ghosh@pine.ece.utexas.edu*)
Dept. of Electr. and Comp. Engineering
The University of Texas at Austin
Austin, Texas 78712, USA

Abstract

Gains due to the parallel execution of Production Systems have been limited by the need for global synchronization before each rule firing. This synchronization can be eliminated by allowing concurrent firing of rules based on the *serializability* criterion for execution correctness. With this enhancement however, the execution demands of the local Rete network within each processor again resurfaces as the performance bottleneck. This paper presents a novel organization that allows the use of multiple functional units to execute the Rete subnetwork within each processor. This organization is reminiscent of current-generation superscalar architectures for numerical processing and indeed borrows some of their synchronizing mechanisms. We present results from a detailed event-driven simulator to quantify the benefits of employing multifunctional units, and show that using a modest number of processors, each containing a small number of functional units, provides the most cost-effective solution for a wide variety of production system benchmarks. A queueing theory based analytical model for predicting the speedup obtained due to the addition of extra functional units in the Rete network, is also developed. To our knowledge, this is the first such analytical model for parallel production system architectures.

keywords: Production Systems, Rete Network, Performance Evaluation, Queueing Theory, Superscalar Processors, Parallel Rule Firing, Matching Engine.

[1]Supported by a grant from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and by Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS) - Brazil.

[2]Supported in part by NSF grant ECS-9307632, ARO contract DAAH 04-94-G0417 and by Faculty Development Awards from TRW Foundation and Schlumberger.

Contents

1 Introduction **1**

2 The Rete Network **3**

3 Base Architectural Model **5**

4 Rete Network with Multiple Functional Units **8**

4.1 Multiple β-Unit Rete Network 9

4.2 Synchronization Issues 10

5 Performance Measurements for Multiple β-Unit Rete Network **11**

5.1 Benchmarking 11

5.2 Simulation Results 12

6 Analytical Model for Multiple β-Unit Rete Network **15**

6.1 Single β-Unit Architecture 15

6.2 Multiple β-Unit Architecture 16

6.3 Simplifications for Analytic Modeling 16

6.4 Single β-Unit Model 17

6.5 Multiple β-Unit Model 21

6.6 Rete Processing Improvement 23

6.7 Analytical Model Predictions 24

7 Concluding Remarks **27**

A Organization and Functioning of the Synchronizing Buffers **29**

A.1 In-Order Buffer 29
A.2 Output Buffer 34

List of Figures

1 Section of a Rete Network 4
2 Processing Element Model 7
3 Rete network with m β-Units 9
4 Synchronizing Buffers in Rete network with m β-Units 10
5 Single β-Unit System 16
6 System with m β-Units 17
7 State Diagram for a Single β-Unit System 18
8 State Diagram for System with m β-Units 22
9 Average Number of Tokens in the System versus λ_β for `moun2` in a Single Processor Architecture 25
10 Prediction of the Rete network speed improvement for `moun2`. 26
11 Actual system level speed improvement due to increased number of β-units for `moun2`. 27
12 Organization of In-Order Buffer (IOB) 31
13 Organization of Output Buffer (OB) 34

List of Tables

1 Static Measures for Benchmarks Used. 11
2 Speedups of a concurrent production system with multiple β-functional units. . . . 13
3 Speedups of a concurrent Production System with multiple β-functional units. . . . 14

4 Parameter Measurements for `moun2` 24

List of Abbreviations and Acronyms

CTSP	Contemporaneous Traveling Salesperson Problem
FIC	Firing Instantiation Controller
FIFO	First-In First-Out
FIM	Firing Instantiation Memory
IOB	In-Order Buffer
OB	Output Buffer
VLSI	Very Large Scale Integration
WME	Working Memory Element

List of Symbols

m	Number of β-units in the Rete Network.
$S_{Rete}(m)$	Speedup of Rete Network with m β-units.
T_α	Average amount of time spent in α-nodes.
$T_\beta(m)$	Average amount of time spent in β-nodes in a Rete Network with m β-units.
β_i	i^{th} β-node in a Rete Network.
R_a	The a-th production in the production set.
P_k	The k-th processor in a multiprocessor machine.
$\mathcal{T}$	Set of all tokens processed in a Rete Network.
T_i	The i^{th} token processed in a β-node.

$\mathcal{A}(T_i)$	Indicates the action of token T_i (add or delete).
$\mathcal{S}(\beta_i)$	Signal of the antecedent associated with β_i.
$\mathcal{C}(\beta_i, \beta_j)$	Indicates possible conflict among tokens directed to β_i and β_j.
h_j	Probability that an α-node produced a bulk of j tokens to deliver to β-node processing.
$H(z)$	Generating function for the h_j.
$\overline{H}$	First moment of h_j.
$\overline{H^2}$	Second moment of h_j.
g_i	Probability that the processing of a token in a β-node produces i new tokens.
$G(z)$	Generating function for the g_i.
$\overline{G}$	First moment of g_i.
$\overline{G^2}$	Second moment of g_i.
λ_β	Arrival rate at β-units.
μ_β	β-unit processing rate.
p_k	Probability that k tokens are present in the β portion of the Rete Network, including the tokens currently being processed.
$P(z)$	Generating function for the p_k.
ρ_1	Utilization rate for a single β-unit Rete Network.
ρ_m	Utilization rate for an m-β-unit Rete Network.
$\overline{N}(m)$	Average number of β-tokens in a Rete Network with m β-units.
$\overline{T}(m)$	Average time to process a β-token in a Rete Network with m β-units.
$I_s(m)$	Improvement in system time due to the use of m β-units instead of a single β-unit in the Rete Network.

1 Introduction

Production systems can encapsulate and execute a variety of rule based or knowledge based systems. When they were first introduced, they were notoriously slow because of the tedious matching of the rule conditions against the entire database required after every rule firing. An early breakthrough in the efficient execution of production systems was made in 1979 by Forgy, who proposed a state saving algorithm called Rete[15]. It was subsequently observed that the matching phase of the Rete algorithm was still very time consuming and a serious bottleneck [17]. This led to a considerable amount of research in the '80s to speed up the execution of production systems, largely by parallelization of the matching step [4, 31]. Unfortunately, the gains observed were quite limited, as one was restricted to the traditional match-select-act cycle in which the next production to be fired is selected from among *all* fireable instantiations, according to a combination of recency and specificity strategies [10].

In industry, production systems have been mostly compiled for execution on serial machines, and increasingly using C or C++ and database languages rather than OPS5. Recently, some products have appeared from Informix, Oracle, Sybase, Microsoft etc., that run on shared memory multiprocessors and/or client-server systems and execute limited forms of production systems. This paper is partly motivated by the renewed interest in parallel production systems due to the recent availability of large (over 100,000 rules) learned production systems [13, 12], and complex querying on large datasets used for example in data mining applications. The need for scalability and faster execution times is resurfacing. Besides faster query matching as performed by database servers, this need can be satisfied by changing the execution model to allow parallel firing of rules and thus avoiding the sequential "match-select-act" cycles.

The notion of parallel rule firing was explored from the mid-80's, with serializability as a new criteria for correctness [22, 23, 36]. This criterion allows a departure from the match-select-act cycle as it results in the elimination of global synchronization before the resolution of the conflict set. In a production system without global synchronization, one can superpose activities in the matching engine with the selection and firing of rules, potentially yielding a significant speed improvement in the parallel execution of production systems. Both Schmolze & Goel [36] and Ishida

et al. [23, 21] proposed parallel execution of rules on distributed memory machines. Rules were partitioned among identical processors and copies of all working memory elements (WMEs) needed for the rules assigned to a given processor were kept in its local memory. Unfortunately, to ensure that the overall execution was serializable, a taxing send-wait protocol was needed, leading to substantial synchronization overheads [36]. Also, a detailed architectural specification and performance evaluation was not made to quantify the advantages and costs of serializability.

Based on the observation that maintaining a consistent view of the overall Rete network's state is similar to keeping caches coherent in shared memory multiprocessors, we have recently proposed a parallel architecture that efficiently executes production systems using a bus-based system with snoopy mechanism [5]. It is seen that global writes are much fewer than reads in typical production systems and thus a bus is not a bottleneck even in systems with tens of processors. Rather, by leveraging technology commercially available and used for small multiprocessors, system-wide synchronization and updates are performed in a few bus cycles. Thus the design is based on ideas from both message-passing and shared memory systems. The architecture also effectively uses small amounts of associative memory to speedup the search for dependencies, executable productions, etc. An eight processor system typically provides a speedup of over a hundred (for reasonably large production systems), as compared to a uniprocessor implementation of Rete with global synchronization.

Interestingly, measurements on this new architecture show that once parallel rule firing is allowed, the matching in the distributed Rete network again becomes the bottleneck. In this paper, we explore the impact of parallel execution of the Rete (sub)-network within each processor by using multiple functional units. As evidenced by recent superscalar processors being produced by almost all leading microprocessor companies [20], it is quite feasible to have multiple functional units within a single chip processor using current VLSI and packaging technology. If the multiple functional units can be usefully employed most of the time and the overheads due to extra dependency checks are not much, then superscalar architectures yield superior performance/cost ratios since the cost of the other components are amortized among the functional units. Of course the economies of scale enjoyed by commodity microprocessors (x86) may overwhelm such a special purpose design as they have done for the vast majority of proposed computer architectures. However we are heartened by

the increasingly lowered costs and shorter turn-around times for special-purpose chips produced by companies such as LSI Logic.

This paper addresses the key question of whether multiple (and identical) functional units can be efficiently used for parallel execution of Rete-based production systems by (i) providing an architectural specification that addresses the synchronization issues raised for correct execution, (ii) performing detailed simulation studies, and (iii)developing an analytical model for predicting and characterizing system behavior. Both measurements of performance obtained from a comprehensive event-driven simulator and predictions from the queueing theory based analytical model, show that a small number of functional units in the Rete network can provide considerable performance improvements.

Sections 2 and 3 provide the needed background by describing the Rete Network, and the base parallel architecture, i.e., one without multiple functional units. Section 4 presents a new organization that permits the implementation of the Rete network with multiple functional units. The synchronization mechanisms used in this organization, detailed in appendix A are somewhat similar to the ones employed in superscalar and superpipelined architectures [25]. Section 5 presents comprehensive performance measurements. Section 6 presents an analytical model to predict the performance improvement expected when the number of functional units is increased.

2 The Rete Network

The Rete Network is a data-flow graph that encodes the antecedents of productions. It also captures the state of a production system in execution by storing the partial matches for every rule. The inputs to the Rete Network are changes to the working memory generated in the act phase of one cycle which are then used to update the match lists. The outputs of the network are changes to the conflict set used to choose a production to be fired in the next cycle. The following discussion of Rete Networks is presented here after Gupta and Forgy [17, 18], Lee and Schor [32], and Miranker [34].

Figure 2 presents a Rete Network encoding the antecedents of two productions, P1 and P2. The network is formed by four different kind of nodes: constant-test nodes, memory nodes, two-input nodes and terminal nodes. In this paper, an information carrying unit that flows through

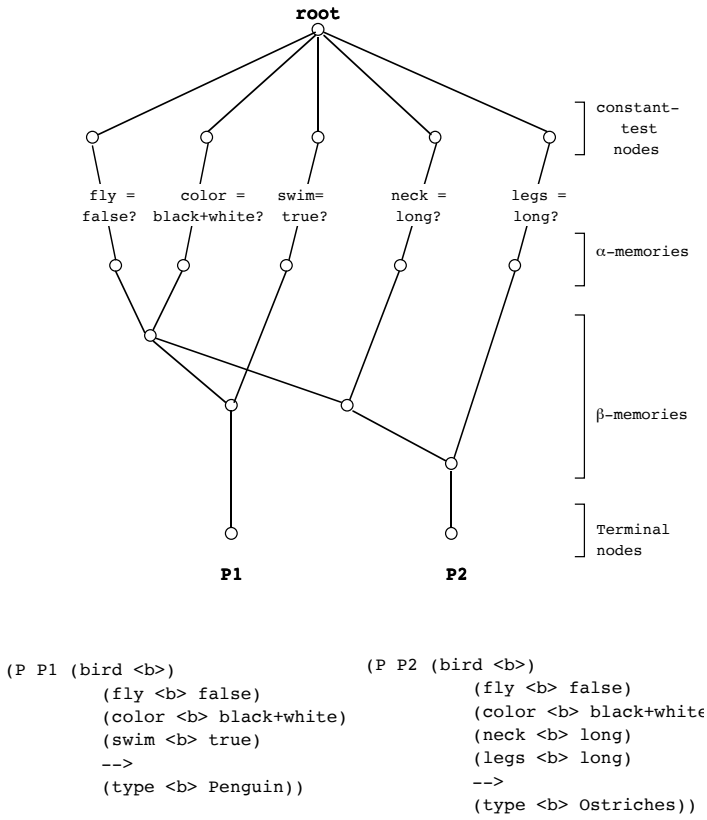

Figure 1: Section of a Rete Network

or is stored at nodes of the Rete network is designated as a "token". Therefore the changes to the working memory arriving at the root network are tokens, the partial matchings stored at the two-input nodes are tokens and the changes to the conflict set that arrive at the terminal nodes are tokens.

The constant-test nodes appear in the first layer of the network. They store attributes that have constants in the value field and perform intra-condition tests to determine if a working memory element satisfies these constant fields of the condition element. In the original Rete Network, the result of this test was stored in a local α-memory [18]. Some authors claim that the α-memories can be eliminated because the constant test takes a negligible amount of time [32]. In this case the one-input nodes are called α-nodes and are memoryless.

Two-input nodes, also called *and*-nodes, *join*-nodes, or β-nodes, perform the matching between distinct condition elements. A β-node has one memory associated with each input. A token arriving

in a β-node come either from another β-node or from an α-node. Whenever a new token arrives in one of a node's inputs, it is compared with all tokens present in the other side memory. Good hashing techniques are necessary to speed up the matching in the β-nodes. Otherwise, it might be necessary to process long lists of tokens [17].

Performance evaluation, modifications, and improvements of Rete are found in the works of Gupta [17, 18], Lee & Schor [32], Stolfo [39], Kelly and Seviora [27, 28], Gaudiot and Sohn [16, 38], and Barachini & Theuretzbacher [8]. Miranker proposed a useful modification to the traditional Rete Network in which intermediate memory nodes are eliminated. This modified Rete is called Treat [34]. A study by Gupta [17] motivated extensive research on concurrent matching. Gaudiot and Sohn [16, 38] proposed a data-driven machine with parallel matching. They advocated the suitability of data-flow machines for the processing of production systems. Kelly and Seviora [27, 28] proposed a multiprocessor architecture that supports comparison level partitioning and consists of a matching multiprocessor attached to a host computer. Rowe *et al.* [9, 35] developed METE/PIPER, an extension of Rete to explore intra-production parallelism in match processing and conflict resolution. Other noteworthy parallel implementations of Rete appear in [1, 19, 31, 37, 39, 40]. Note that none of these works are based on parallel and serializable rule firing, which leads to qualitatively different characteristics of Rete execution.

3 Base Architectural Model

This section summarizes the base architectural model in which each processor of the parallel machine has a single functional unit. Detailed description of the model, along with proof of correctness and performance evaluation, can be found in [2, 5, 7][3]. For example, with 8 processors, this base model is able to provide a speedup factor in the tens or hundreds (depending on the production system benchmark) over a single processor that does not use associative memories or allow parallel rule firing.

[3]The paper "A Concurrent architecture for serializable production systems" is currently under review for publication in the *IEEE transactions on Parallel and Distributed Processing*. It can be download by anonymous ftp from `pegasus.ece.utexas.edu` in `/pub/papers/concurps_itpdp.ps.Z`.

Our architectural model consists of a moderate number of identical processors interconnected through a Broadcasting Interconnection Network, such as a bus. Each of the processors has the internal organization shown in Figure 2. An I/O processor attached to the bus 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 based on the firing frequency of each production measured in previous runs of the same problem [3]. A processor reads data only from its local memory, i.e., no read operations are performed over the network. Due to the absence of network reads and the low frequency of network writes, a simple bus should be adequate as the broadcasting system. This conclusion is supported by detailed experimental results showing the bus not to be a bottleneck even for a twenty processor system.

A number of associative memories implement a system of lookaside tables to allow parallel operations within each processor. This scheme does not allow parallel production firing within a processor, but allows the match-select-act phases of a production system to overlap. A snooping directory isolates the activities in remote processors from the activities in a local processor, and interrupts a local operation only when pieces of data that affect the local processor are broadcast over the network.

All tokens propagated over the bus 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 to verify which previously enabled productions are now disabled; remove such productions from the Fireable Instantiation Memory (FIM) along with their antecedents; and 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 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[4],

[4]Low overhead in token removing is the most salient advantage of the Treat algorithm [34].

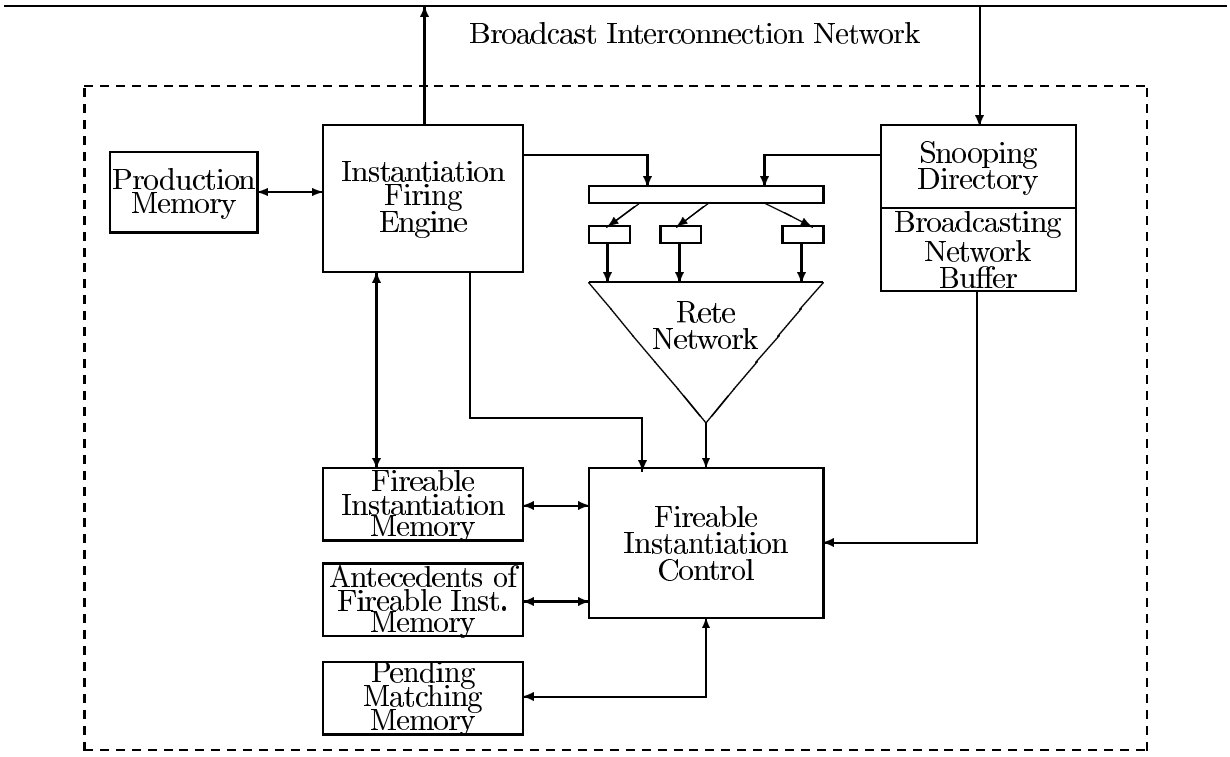

Figure 2: Processing Element Model

and allows the maintenance of the β-memories of the original Rete algorithm [15]. This combination of the advantages of Rete and Treat is made possible by the storage of negated conditions in the representation of fireable instantiations 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. Local productions are further classified into two categories depending on whether a certain production can start firing at any time as long as its antecedents are satisfied, or whether it needs to wait if a related production is being currently fired in a remote processor. An execution model that guarantees non-serializable behavior with very little synchronization, has been developed based on the above categorization of productions [7].

To select a production to fire, the Instantiation Firing Engine performs an associative search in FIM to find the most recently enabled production. If the selected production is remote, the engine places a request for ownership of the bus. Upon receiving bus ownership, the engine waits until

all outstanding tokens from previous broadcastings are processed by FIC. The engine accesses FIM to verify whether the selected production is still fireable. If so, it proceeds to execute its actions, propagating tokens that change shared WMEs in bus and sending tokens that modify only local WMEs to FIC and Rete.

The Snooping Directory 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. The Snooping Directory "snoops" the bus and captures 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, and is processed as soon as the local production processing finishes.

The Pending Matching Memory is necessary to store tokens that are in the Rete network. Whenever a change to the conflict set[5] is generated in the Rete network, FIC performs an associative search in the Pending Match Memory to verify if a later modification invalidates such change. This mechanism prevents races between the Instantiation Firing Engine and Rete.

4 Rete Network with Multiple Functional Units

The architectural model presented in section 3 assumed that the matching was performed by a Rete network with a single functional unit. A closer look at Production System execution on this model reveals that the bottleneck is in the processing of tokens in the β-nodes of the Rete network rather than at the associative memories or in the communication bus. This motivates the use of a multiple functional unit Rete network in the architecture presented in Figure 2. We now consider the use of a single functional unit for α-node matching and one or more functional units to process tokens in the β-nodes.

The use of multiple β-units adds complexity to the problem of synchronizing the matching of tokens in the Rete network with the firing of productions in the Instantiation Firing Engine. To ensure a correct operation of the multiple β-unit architecture, two synchronizing structures are added: an *In-Order Buffer* and an *Output Buffer*. This section presents the organization for this

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

new Rete network scheme.

4.1 Multiple β-Unit Rete Network

In the organization presented in Figure 3, the α-unit is responsible for processing tokens in α-nodes. Each incoming token is sequentially tested against all α-nodes of the network. Whenever there is a match with an α-node, the α-unit produces one copy of the token for each one of the β-nodes that are successors of the matching α-node. Before forwarding these new tokens to the β-units, the α-unit attaches to each one the identity of one of the successor β-nodes. Therefore the processing of a single token in the α-unit may result in the generation of many tokens to be inserted in the *external* β-queue. Note that β-units also share an *internal* input queue for tokens generated by β-nodes.

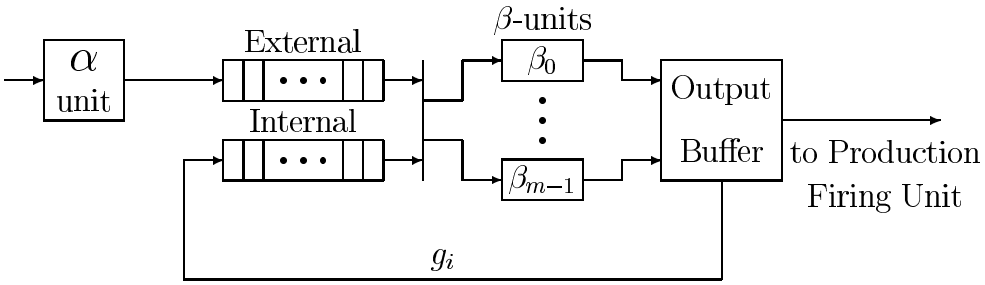

Figure 3: Rete network with m β-Units

To reduce the average time that tokens spend in the Rete network, a token placed in the external queue is processed only when the internal queue is empty. A free β-unit will take the first token from the queue, read the memory locations corresponding to the β-node to which the token is destined, execute all the β-tests and then place the newly generated tokens at the end of the internal queue. When a token destined for a P-node is produced by a β-unit it is forwarded to the production firing unit.

This study does not consider the use of multiple α-units. Therefore, the amount of speedup in the Rete network, $S_{Rete}(m)$, according to Amdahl's Law [20], is limited by the amount of time

spent in the non-parallelized portion of the algorithm.

$$S_{Rete}(m) = \frac{T_\alpha + T_\beta(1)}{T_\alpha + T_\beta(m)}, \tag{1}$$

where m is the number of β-units in the architecture, T_α is the average amount of time spent in α nodes, and $T_\beta(m)$ is the average amount of time spent in β nodes when m identical β-units are used. Assuming that the processing time at the β-units is very short when a large number of units is available, the maximum speedup that can be obtained with such an architecture is given by

$$\lim_{m\to\infty} S_{Rete}(m) = 1 + \frac{T_\beta(1)}{T_\alpha}. \tag{2}$$

4.2 Synchronization Issues

Because multiple tokens are simultaneously processed in the β-units, the order in which the results appear at the output of these units needs careful consideration. The organization presented in Figure 3 allows out-of-order execution of tokens in the β-units and might cause token deletions to be processed before the corresponding additions. Synchronizing mechanisms are necessary to ensure that out of order processing of tokens does not lead to wrong results. Simple solutions such as allowing only tokens originated from a single external token to proceed to the β-units are too conservative and might offset the advantages of a multiple β-unit organization. It is necessary to create a synchronization mechanism that imposes minimum overhead and guarantee correct results in every situation.

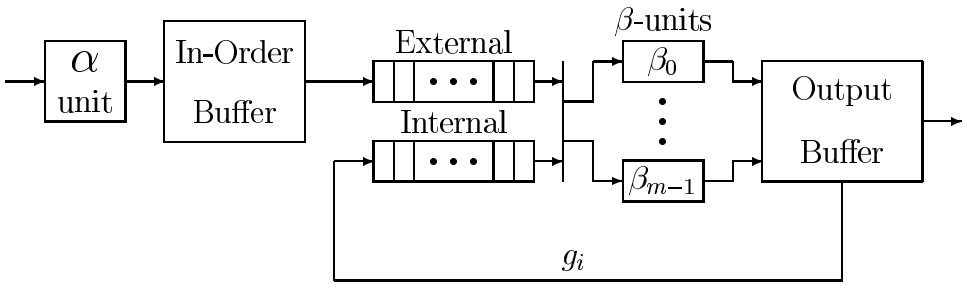

Figure 4: Synchronizing Buffers in Rete network with m β-Units

The solution shown in Figure 4 uses an *In-Order Buffer* (IOB) and an *Output Buffer* (OB). IOB prevents conflicting tokens from entering the β-unit queues simultaneously. OB detects when

a token is leaving the β-units and reports it to IOB. A detailed description of these two buffers is given in Appendix A

5 Performance Measurements for Multiple β-Unit Rete Network

Performance evaluation can be accomplished through measurement, simulation, and analytic modeling [26]. Measurement consists of observing actual values for specified parameters in an existing system. Simulation consists in creating a model for the behavior of a system, writing a computer program that reproduces this behavior, feeding the program with an appropriate sample of the workload of the actual system, and computing selected parameters of interest. In analytic modeling a mathematical model of the system is created and its solution provides the performance evaluation. The problem with analytic modeling is that few detailed mathematical models can actually be solved. The good news is that even simplified models often deliver remarkably good approximations for performance estimates [26]. In this section we provide simulation results, and then develop an analytical model in section 6.

5.1 Benchmarking

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.

Like in numerical processors, the performance of a production system machine is highly dependent on the production system program that it executes. Unfortunately, a well known weakness of production system machine research is the lack of a comprehensive and broadly used set of benchmarks for evaluation of performance. To evaluate the performance of the multiple functional unit Rete Network, we used three benchmarks obtained from other researchers and developed a new benchmark in which the number of productions and the database size can be independently changed to allow researchers to study various aspects of new architectures. This new benchmarking, called Contemporaneous Traveling Salesperson Problem (CTSP), is presented in detail in [2] and [6].

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 — 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 fifteen cities per country. Our solution to the constraint *Confusion of Patents Problem* presented in [14, 24] is `patents`. 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. 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. All these benchmarks are detailed in [2].

5.2 Simulation Results

To evaluate the speed improvement obtained from the combination of inter-production parallelism and matching performed by a multiple functional unit Rete network, we developed a comprehensive event-driven system level simulator that takes as an input an OPS5-like production system program, and produces measurements of the time necessary to solve the problem specified by the program. The OPS5 parser and the encoding of productions in an internal data structure was developed

Bench.	# Proc	# of Beta Units						
		1	2	3	5	7	8	10
patents	1	1.00	1.98	2.88	4.64	6.32	6.93	8.44
	2	1.54	3.02	4.50	7.30	9.78	10.62	13.09
	5	2.70	5.28	8.03	13.45	18.71	20.83	23.48
	10	5.34	10.56	14.79	21.28	23.63	24.07	24.55
	15	6.20	12.82	18.55	22.07	23.40	23.79	24.49
	20	7.05	14.23	19.45	22.22	23.49	23.98	24.61
waltz2	1	1.00	1.99	2.87	4.03	4.80	5.05	5.42
	2	1.73	3.17	4.05	5.07	5.61	5.78	6.03
	5	3.46	4.64	5.33	5.99	6.33	6.44	6.55
	10	3.46	4.64	5.31	5.99	6.35	6.43	6.55
hotel	1	1.00	1.32	1.36	1.36	1.36	1.36	1.36
	2	1.73	1.78	1.79	1.80	1.80	1.80	1.81
	5	3.70	3.77	3.75	3.77	3.79	3.79	3.79
	10	5.87	6.96	6.64	6.96	6.76	6.90	6.83
	15	5.95	7.17	7.16	7.22	7.25	7.30	7.28
	20	5.95	9.15	8.95	9.00	8.82	9.08	8.87

Table 2: Speedups of a concurrent production system with multiple β-functional units.

by Anurag Acharya from Carnegie Mellon University. This simulator is also capable of providing information about the time spent in each one of the subsystems and the size of the associative memory necessary to implement the system.

Tables 2 and 3 present the speedups obtained by increasing the number of processors and the number of β-units in the architecture of Figure 2. All measures presented represent architectures with a separate unit for α-node processing. On the top of each column is the number of β-units in the architecture. These results clearly show the advantages of using multiple β-units. In general, given limited resources, it is preferable to use fewer, more powerful processors than a larger number of less powerful ones. For example, a two processor system with five β-units per processor performs better than a five processor system with two β-units per processor.

There are few situations in which increasing the number of β-units or the number of processors (or both) results in a reduction of speedup. This happens because: (1) the architecture utilizes

Bench.	# Proc	# of Beta Units						
		1	2	3	5	7	8	10
south	1	1.00	1.97	2.89	4.63	6.00	6.80	7.99
	2	1.70	3.26	4.76	7.25	9.20	10.24	11.65
	5	3.07	6.06	9.94	11.88	15.91	16.05	18.38
	10	3.41	6.71	10.05	15.63	18.42	19.58	20.73
	15	4.22	8.03	11.29	16.07	18.77	20.54	21.21
	20	4.38	8.50	10.32	15.12	20.80	21.60	22.27
south2	1	1.00	1.96	2.86	4.34	5.62	6.20	7.12
	2	1.75	3.43	4.84	6.69	8.08	8.66	9.94
	5	3.06	5.59	7.73	10.85	12.70	13.03	13.75
	10	4.21	7.60	10.22	12.87	14.75	15.06	15.03
	15	4.45	8.33	11.07	14.94	17.12	16.49	18.05
	20	4.58	8.44	11.48	15.27	17.14	17.43	17.93
moun2	1	1.00	1.94	2.81	4.40	5.82	6.44	7.58
	2	1.85	3.41	4.97	7.31	9.11	9.87	10.90
	5	3.43	6.53	8.93	12.67	15.03	15.88	16.29
	10	5.02	9.13	12.30	16.42	19.13	19.89	20.23
	20	7.24	12.98	18.18	25.84	30.43	32.09	33.91

Table 3: Speedups of a concurrent Production System with multiple β-functional units.

a partially informed selection mechanism that might result in the firing of extra productions; (2) for a given configuration in an specific benchmark, the firing engine might select productions and wait for the possession of the bus just to find out that the selected production is no longer fireable and needs to be aborted. Therefore, Tables 2 and 3 should be examined for the trends with the increasing of the number of processors and β-units, and not for any specific value.

The higher speedup delivered by a multiple functional unit Rete network confirms some early observations in production system research that the matching phase of the execution constitutes a bottleneck. However, for some benchmarks, the improvement obtained by a faster Rete network saturates with a rather small number of functional units, evidencing the need for multiple processors.

6 Analytical Model for Multiple β-Unit Rete Network

Analytical modeling is a powerful and underused tool in the study of production system machine performance. Yukawa *et al.* [43, 29] construct an analytic model for the performance of Rete network based on basic notions of probability, utilizing a "back of the envelope" method similar to the one largely employed by Cragon [11] for the analysis of superpipelined and superscalar processors. Wang et al. [41] constructed an analytical model to measure performance of parallel-rule firing production systems based on a transaction model.

In this research we use what Kant [26] describes as an "hybrid modeling" method. We developed an event-driven simulator to obtain the parameters necessary to evaluate the performance of the Rete network with multiple β-units described in section 4. Here, we construct an analytic model for the processing of tokens in the Rete network which uses some of these parameters. This model allows the estimation of the improvement in the speed of the Rete network when m β-units are used.

For clarity, we first present a single β-unit system, introducing the generating function technique that is used to solve the analytic model [42, 30, 26]. In section 6.5 we develop a model for a system with m servers, and verify that it reduces to the single server model when $m = 1$.

6.1 Single β-Unit Architecture

Consider an organization with a single β-unit and a single α-unit. A number of simplifications are necessary to construct the analytical model for the Rete network with multiple functional units. Delays due to the synchronizing *In-Order Buffer* are not considered in the analytical model. Also we consider an organization with a single queue for both external and internal tokens. Figure 5 shows the single β-unit organization considered for the analytical model.

Whenever a token matches an α-node while being processed in the α-unit, a number of tokens are produced to be delivered to the β-FIFO. This group of tokens is called a *bulk*. The probability that a bulk has j tokens is measured by h_j. The model does not consider empty bulks, i.e., an arrival occurs when at least one token arrives in the β-FIFO. Therefore $h_0 = 0$.

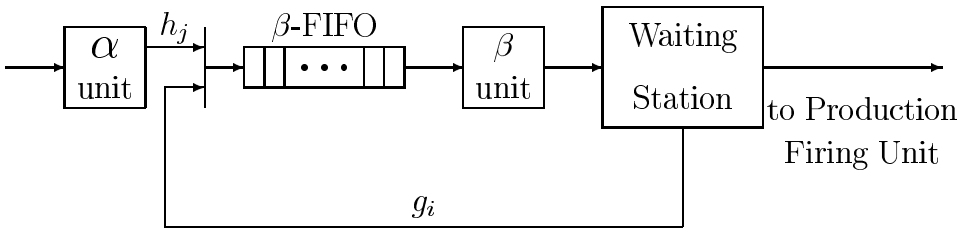

Figure 5: Single β-Unit System

Whenever free, the β-unit will take the first token from the queue, read the memory locations corresponding to the β-node to which the token is destined, execute all the β-tests and then place the newly generated tokens at the end of the β-FIFO. If the processing of a token in a given β-node produces a new tokens, and that β-node has b β-node successors, we consider that ab new tokens were generated and placed at the end of the queue. The probability that i new tokens are generated when a token is processed in a β-node is given by g_i. Also, any new instantiation at a terminal node is sent to the production firing unit that will select one (or more) production to be fired.

For the construction of the analytical model, we consider that there is a "waiting station" at the β-unit output that allows it to hold all the tokens generated until the end of the execution of the current token, and then adds all of them at once to the queue. Although this assumption may not reflect the actual behavior of a physical machine[6], it simplifies the analytical model.

6.2 Multiple β-Unit Architecture

The organization considered for the construction of an analytical model for the Rete network with multiple β-units is shown in Figure 6. The only difference with the single β-unit architecture is the number of β-units used to process tokens in β-nodes.

6.3 Simplifications for Analytic Modeling

In this model, the arrival of a bulk of tokens in the β-queue from an α-node is called an *external arrival*. The arrival of a group of tokens from the *waiting station* into the β-FIFO is an *internal*

[6]In an actual machine the β-unit would place tokens in the queue as they are produced.

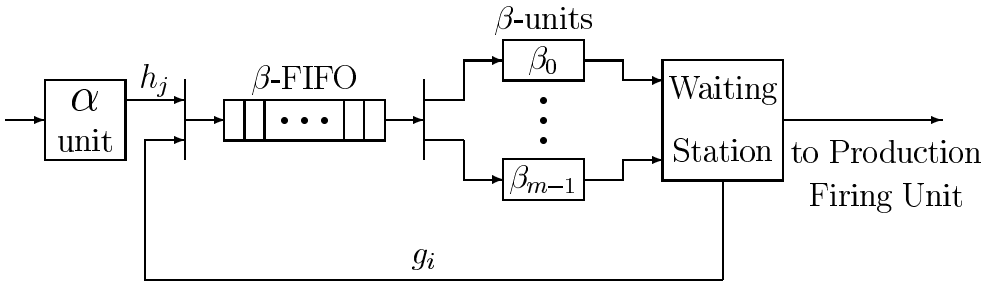

Figure 6: System with m β-Units

arrival. The moment at which a β-unit finishes processing a token and places all the newly generated tokens (if any) in the β-FIFO is called *departure*. To develop the queueing theory model, we make the following assumptions:

- The β-FIFO has infinite capacity.
- The overhead of placing tokens in the β-FIFO is zero.
- The external arrival of tokens is a Poisson process.
- The processing time for tokens in the β-nodes follows a Poisson process.

The main problem with the above assumptions is that events that occur in the machine are not independent of each other. In fact they not only depend on the specific production system being executed, but also change as the execution of it proceeds. However, the simplified model obtained still produces a good estimate for the amount of time that a token spends in the system, and thus for the reduction in the average time spent by a token in an m β-unit system compared with a single β-unit system.

6.4 Single β-Unit Model

The infinite Markov chain that represents the single β-unit system is presented as a state-transition-rate diagram in Figure 7. The circles represent states and the arcs represent state transitions. The value associated with each arc indicates the transition rate. In this representation, the system is

Figure 7: State Diagram for a Single β-Unit System

We are interested in the steady state behavior of the system. At steady state, the input flow

must equal the output flow in each state. Therefore, we can write difference equations for the system.

$$(\lambda_\beta \, + \, \mu_\beta \sum_{i=0}^{\infty} g_i) \, p_k \, = \, \mu_\beta \, g_0 \, p_{k+1} \, + \, \lambda_\beta \sum_{j=0}^{k-1} p_j \, h_{k-j} \, + \, \mu_\beta \sum_{i=1}^{k} p_i \, g_{k-i+1}, \tag{3}$$

where p_k is the probability of k tokens being in the system, including the token being processed.

By definition, the sum of all probabilities g_i is equal to unity. Using $\sum_{i=0}^{\infty} g_i \, = \, 1$, equation 3 can be written as follows:

$$(\lambda_\beta \, + \, \mu_\beta) \, p_k \, = \, \mu_\beta \, g_0 \, p_{k+1} \, + \, \lambda_\beta \sum_{j=0}^{k-1} p_j \, h_{k-j} \, + \, \mu_\beta \sum_{i=1}^{k} p_i \, g_{k-i+1}. \tag{4}$$

Examining the transitions into and out of state 0, we can write the boundary equation 5.

$$\lambda_\beta \sum_{j=0}^{\infty} h_j \, p_0 \, = \, \mu_\beta \, g_0 \, p_1 \tag{5}$$

But the sum of all probabilities h_j is also equal to unity. Therefore the boundary equation can be simplified to equation 6

$$\lambda_\beta \, p_0 \, = \, \mu_\beta \, g_0 \, p_1 \tag{6}$$

From equations 4 and 6, using Z-transforms we obtain the generating function for the system with a single β-unit [2]:

$$P(z) \, = \, \frac{\mu_\beta \, (1 \, - \, \rho_1) \, (z \, - \, G(z))}{\lambda_\beta \, z \, [1 \, - \, H(z)] \, + \, \mu_\beta \, [z \, - \, G(z)]}, \tag{7}$$

where $P(z)$, $G(z)$, $H(z)$, and ρ_1 are defined as

$$P(z) \, = \, \sum_{k=0}^{\infty} p_k \, z^k, \tag{8}$$

$$G(z) \, = \, \sum_{k=0}^{\infty} g_k \, z^k. \tag{9}$$

$$H(z) \, = \, \sum_{k=0}^{\infty} h_k \, z^k. \tag{10}$$

$$\rho_1 \, = \, \frac{\rho_0 \, \overline{H}}{(1 \, - \, \overline{G})} \, = \, \frac{\lambda_\beta \, \overline{H}}{\mu_\beta \, (1 \, - \, \overline{G})}, \tag{11}$$

$$\overline{G} \, = \, \lim_{z \to 1} G^{(1)}(z) \, = \, \sum_{i=0}^{\infty} i \, g_i \tag{12}$$

$$\overline{H} \, = \, \lim_{z \to 1} H^{(1)}(z) \, = \, \sum_{i=0}^{\infty} i \, h_i \tag{13}$$

$G^{(1)}(z)$ and $H^{(1)}(z)$ are the first derivative of $G(z)$ and $H(z)$ with respect to z.

The *utilization factor* ρ_0 is a ratio between the external arrival rate λ_β and the β-unit processing rate μ_β; ρ_0 represents the utilization rate in an M/M/1 system, i.e., a system with no bulk external arrivals and where new tokens are not generated at the β-units. The value ρ_1 is the utilization rate of a single β-unit system with bulk arrivals and "feedback", i.e., a system in which the β-units generate new tokens. This rate must be positive, therefore the *first moment* of g_i, $\overline{G}$ must be less than one. Also, for stability, we must have $\rho_1 < 1$ what implies that

$$\frac{\lambda_\beta}{\mu_\beta} < \frac{(1-\overline{G})}{\overline{H}} \tag{14}$$

¿From the definition in equation 8 it follows that

$$\lim_{z\to 1} P^{(1)}(z) \;=\; \lim_{z\to 1} \sum_{k=1}^{\infty} k\, p_k\, z^{k-1} \;=\; \sum_{k=1}^{\infty} k\, p_k \;=\; \overline{N}(1), \tag{15}$$

where $P^{(1)}(z)$ represents the first derivative of $P(z)$ with respect to z, and $\overline{N}(1)$ is the average number of tokens in the single β-unit system including the token currently being processed.

For g_i we verify that

$$\lim_{z\to 1} G^{(2)}(z) \;=\; \lim_{z\to 1} \sum_{i=0}^{\infty} i\,(i-1)\, g_i\, z^{i-2} \;=\; \sum_{i=1}^{\infty} i\,(i-1)\, g_i \;=$$
$$\sum_{i=1}^{\infty} i^2\, g_i \;-\; \sum_{i=1}^{\infty} i\, g_i \;=\; \overline{G^2} \;-\; \overline{G}, \tag{16}$$

where $G^{(2)}(z)$ is the second derivative of $G(z)$ with respect to z, and $\overline{G^2}$ is called the *second moment* of $G(z)$. This indicates that the limit as z goes to 1 of the second derivative of $G(z)$ is equal the difference between the second and the first moments of $G(z)$.

In a similar fashion, we can establish that

$$\lim_{z\to 1} H^{(2)}(z) \;=\; \overline{H^2} \;-\; \overline{H}. \tag{17}$$

Calculating the derivative of expression 7, taking the limit as z goes to 1, and using the results 12, 13, 15, 16, and 17, we obtain the following expression for $\overline{N}(1)$ in terms of the moments of $G(z)$ and $H(z)$ [2].

$$\overline{N}(1) \;=\; \frac{\rho_1}{2\,(1-\rho_1)}\left[\frac{(\overline{H^2}+\overline{H})}{\overline{H}} \;+\; \frac{(\overline{G^2}-\overline{G})}{1-\overline{G}}\right] \tag{18}$$

Since the values that appear on the right hand side of equation 18 can be measured in the simulator developed, we can use this equation to estimate the average number of tokens in the single β-unit Rete network. We use Little's result to obtain the average time that a token spends in the system [33, 30].

$$\overline{T}(1) \;=\; \frac{\overline{N}(1)(1-\overline{G})}{\lambda_\beta\,\overline{H}}, \tag{19}$$

where $\overline{T}(1)$ represents the average total time that a token spends in a single β-unit system, including the time the token is waiting in the queue and the processing time. Observe that we have to consider the total arrival rate in the β-FIFO, i.e., external tokens originated in the α-nodes as well as tokens generated in the β-nodes. In equation 19, the external arrival rate is $\lambda_\beta\,\overline{H}$ and the arrival rate due to generation of tokens in β-units is $(1-\overline{G})^{-1}$.

6.5 Multiple β-Unit Model

In this section we are interested in estimating the average time that a token spends in the m β-units organization shown in Figure 6.

Figure 8 presents the state-transition-rate diagram for the system with m β-units. The main difference from the single β-unit model is that the transition rate out of states 1 through $m-1$ due to the processing of tokens in β-units is proportional to the number of tokens in the system. This occurs because if $i<m$ tokens are present, all tokens are being processed, and $m-i$ β-units are idle. Therefore, the processing rate is proportional to the number of tokens in the system. If, however, $j\geq m$ tokens are present in the system, m tokens are being processed and $j-m$ tokens are waiting in the queue; therefore, the processing rate is proportional to the number of β-units. Every state (except for state 0) has a self transition. If $i<m$, the self transition rate for state i is $i\,\mu_\beta\,g_1$. Otherwise the self transition rate is $m\,\mu_\beta\,g_1$.

Figure 8: State Diagram for System with m β-Units

The m–1 boundary conditions of this system are expressed by equation 20. The flow conservation equation for state $k \geq m$ is given by equation 21.

For $k < m$:

$$(\lambda_\beta \, + \, k\,\mu_\beta)\, p_k \; = \; (k \, + \, 1)\,\mu_\beta\, g_0\, p_{k+1} \, + \, \lambda_\beta \sum_{j=0}^{k-1} p_j\, h_j \, + \, \mu_\beta \sum_{i=1}^{k} i\, p_i\, g_{k-i+1}. \tag{20}$$

For $k \geq m$:

$$\begin{aligned}(\lambda_\beta \, + \, m\,\mu_\beta)\, p_k \quad = \quad & m\,\mu_\beta\, g_0\, p_{k+1} \, + \, \lambda_\beta \sum_{l=0}^{k-1} p_l\, h_l \, + \\ & \mu_\beta \sum_{i=1}^{m-1} i\, p_i\, g_{k-i+1} \, + \\ & \mu_\beta \sum_{j=m}^{k} m\, p_j\, g_{k-j+1}.\end{aligned} \tag{21}$$

To compute the generating function $P(z)$, we have to sum equation 20 from 0 to m–1, and sum equation 21 from m to infinity, and then combine both results and simplify [2]. This results in

$$P(z) = \frac{\mu_\beta\,[z - G(z)]\;\sum_{k=0}^{m-1}(m-k)\,p_k\,z^k}{\lambda_\beta\, z\,[1 - H(z)] \, + \, m\,\mu_\beta\,[z - G(z)]}. \tag{22}$$

Computing the limit of $P(z)$ as z goes to 1, and making the result equal to 1 (series sum property), we obtain relation 23. Equation 20 provides m equations and $m+1$ unknowns. The relation expressed in 23 is the last equation we need to solve this linear system and obtain the values for the boundary probabilities p_0, p_1, ..., p_m.

$$\sum_{k=0}^{m-1} (m-k)\, p_k \;=\; m\,(1-\rho_m) \tag{23}$$

$$\rho_m \;=\; \frac{\lambda_\beta\, \overline{H}}{m\,\mu_\beta\,(1-\overline{G})} \tag{24}$$

To guarantee stability, the utilization factor ρ_m for the system with m β-units must be less than unity. Therefore the distributions of g_i and h_i must have the following property:

$$\frac{\overline{H}}{1-\overline{G}} < \frac{\lambda_\beta}{m\,\mu_\beta}. \tag{25}$$

The average number of tokens in the system with m β-units, including the tokens that are currently being processed, is obtained by computing the limit of the first derivative of $P(z)$ as z goes to 1 [2].

$$\overline{N}(m) \;=\; \frac{\rho_m}{2\,(1-\rho_m)} \left[\frac{(\overline{H^2}+\overline{H})}{\overline{H}} \;+\; \frac{(\overline{G^2}-\overline{G})}{1-\overline{G}} \right] \;+\; \frac{\sum_{k=0}^{m-1} k\,(m-k)\,p_k}{m\,(1-\rho_m)}. \tag{26}$$

Note that expression 26 is identical to expression 18 when $m = 1$, as expected. The first and second moments of $G(z)$ and $H(z)$ are obtained from measures of g_i and h_j in the simulator. The boundary probabilities are obtained by solving the system of linear equations mentioned previously.

6.6 Rete Processing Improvement

Using Little's result, the ratio between the average time spent in the single β-unit system and the average time spent in the system with m β-units can be obtained from the average number of tokens in each one of these systems. We define the improvement in system time according to equation 27.

$$I_s(m) = \frac{\overline{T}(1)}{\overline{T}(m)} = \frac{\overline{N}(1)}{\overline{N}(m)}, \tag{27}$$

assuming that the effective arrival rate of the system does not change with the number of β-units. $I_s(m)$ indicates how much faster a token goes through the β-node portion of the Rete network when m β-units are used instead of one.

We purposely avoid calling this performance improvement *speedup* because of the loaded meaning of that word. Speedup is a system level concept that is applied to different settings, e.g. fixed-time speedup, fixed-load speedup, memory-bound speedup, etc [20]. Also, except for very rare cases, the speedup of a computer system is always a sublinear function of the number of processors in the system. In our system, because of the nature of the incoming flow of tokens, the amount of improvement in the time spent in the β-network can be superlinear. This happens because the existence of occasional bursts of traffic in the incoming flow of tokens can cause tokens to spend considerable amount of time waiting in the β-unit input queue. Of course, if the incoming flow of token were to be steady, the improvement in the time spent in the system would be at most linear.

6.7 Analytical Model Predictions

The analytical model presented in this paper assumes a steady state in the flow of tokens through the Rete network. An approximation of such a situation is only encountered in fairly large production system programs. We present measurements for the benchmark `moun2`, which is the largest benchmark presented in Section 5. Table 4 presents measurements for the first and second moment of g_i and h_j, for the service rate μ_β and the average number of tokens in the single β-node system $\overline{N}(1)$ as measured in an event driven simulator.

# Proc	$\overline{G}$	$\overline{G^2}$	$\overline{H}$	$\overline{H^2}$	μ_β	$\overline{N}(1)$
1	0.78	3.35	5.45	40.22	0.00096	644.8
2	0.80	3.10	2.89	11.58	0.00093	244.4
4	0.79	4.17	2.15	6.15	0.00096	160.2
6	0.80	4.20	1.59	3.51	0.00094	89.6
10	0.80	5.06	1.41	2.48	0.00102	47.8

Table 4: Parameter Measurements for `moun2`.

In the architecture presented in section 3, the production set is divided among the processors in

the machine. Therefore, a machine with a larger number of processors has smaller Rete networks in each processor. Moreover, the amount of tokens processed in each Rete network is smaller. Note that the first and second moments of g_i do not change significantly when the Rete network is divided into a larger number of networks, but rather seems to be a characteristic of the benchmark program. On the other hand, the moments of h_j do change substantially when each processor has smaller Rete networks because in such networks each α-node has fewer successors. The value of μ_β is also independent of the size of the Rete network. The measures presented in Table 4 were obtained by forcing the simulator to implement a single β-FIFO to replicate the simplification used in the analytical model.

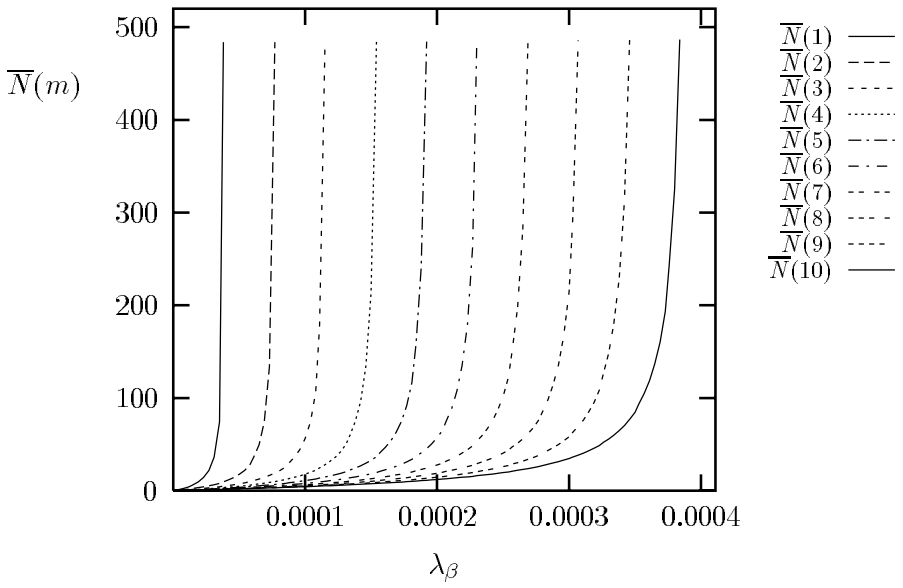

Figure 9: Average Number of Tokens in the System versus λ_β for `moun2` in a Single Processor Architecture

Figure 9 shows the variation in the average number of tokens in the system with the value of the arrival rate λ_β for Rete network of *moun*2, in a single processor architecture with one up to ten β-units. Notice that there is a dramatic increase in the number of tokens in the system when the utilization rate tends to one. For systems operating close to such an asymptote, the addition of a single β-unit can improve the performance significantly. This improvement is due to the reduction in the amount of time spent waiting in the queues.

A difficulty in the utilization of this analytical model to estimate performance improvements is the correct estimation of arrival rate λ_β. We can obtain the arrival rate for a single β-unit system from direct measurement in the simulator. However, this rate does not remains constant when the number of β-units is increased. The Instantiation Firing Engine works as an outer loop that receives new instantiations from the output of the Rete network and generates new actions that are placed in the Rete network input. A Rete network with a larger number of β-units produces changes to the conflict set at a faster pace resulting in a higher arrival rate.

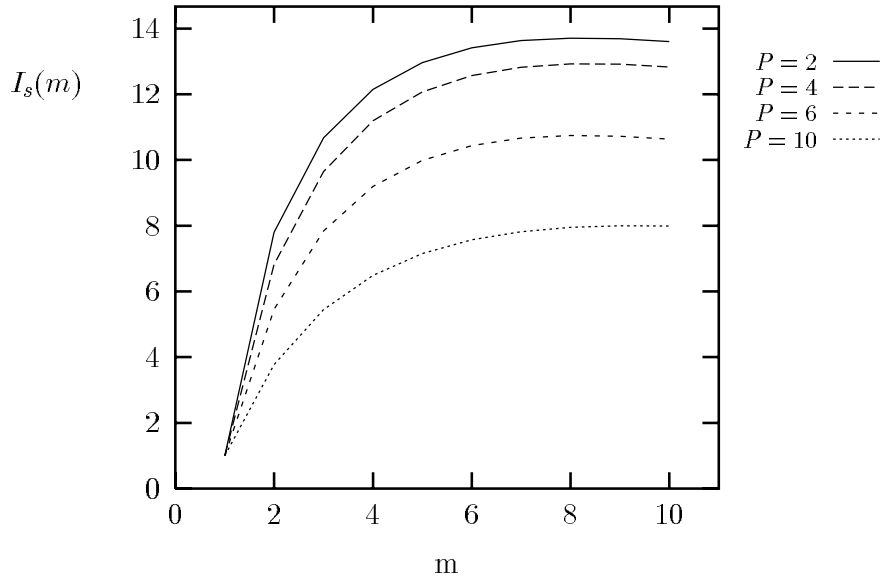

Figure 10: Prediction of the Rete network speed improvement for `moun2`.

The simulator developed has the capacity to simulate multiple β-unit Rete networks. Therefore, we can measure the amount of change in the arrival rate λ_β when the number of β-units is increased. Using the measured λ_β values, we obtain the speedup curves shown in Figure 10. Future research with experimental measurements of a larger set of benchmarks might determine a general rule to estimate the variation of the arrival rate with the number of β-units.

Figure 11 presents the actual speed improvement obtained from measures in the simulator when multiple β-units are used in the architecture. Notice that these curves only show the speed improvement obtained due to the addition of multiple β-units *after* the partition of the productions among

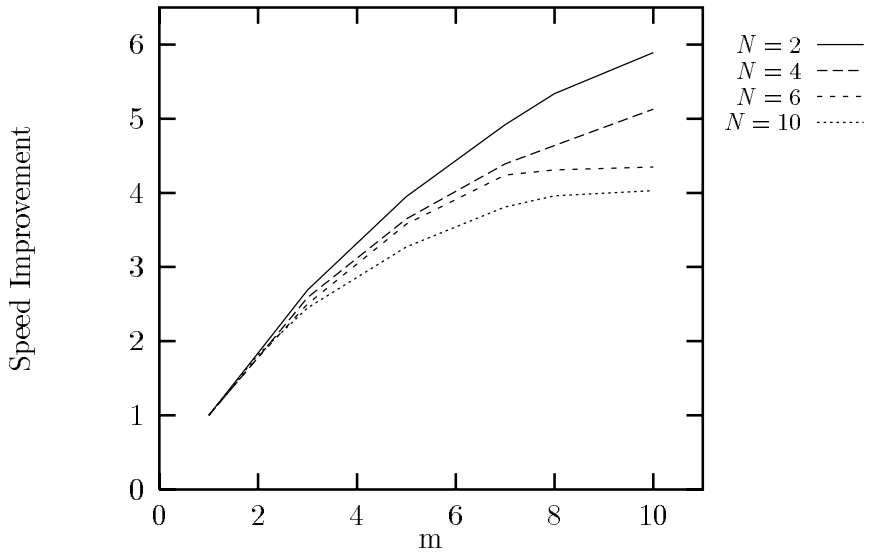

Figure 11: Actual system level speed improvement due to increased number of β-units for `moun2`.

multiple processors. As the graphic shows, when fewer processors are used in the architecture the Rete network is heavy loaded and thus multiple functional units produce more speed improvement. The base of comparison for each curve in Figure 11 is an architecture with the same number of processors and a single β-unit as base of comparison. Because Figure 10 plots the predicted speed improvement only in the Rete network while Figure 11 plots the actual speed improvement for the whole machine, the absolute values are quite different. However, comparing the plots of Figures 10 and 11 we verify that the analytical model was successful in anticipating the trend of these curves. A computer architect could estimate the number of β-units to use in a design only from the analytical model predictions.

7 Concluding Remarks

1

Recent growth in expert databases, large rule bases, intelligent transaction processing and data mining applications have led to a renewed demand for executing large production systems in (near) real time. This paper is an initial step in addressing this need by providing an innovative organiza-

tion that allows for considerable speed gains in the implementation of Rete networks. The multiple functional unit Rete matching engine adapts superscalar processor design concepts for fast and correct execution of production systems.

The proposed matching engine was used in a parallel architecture that produces correct results according to the serializability criterion. This architecture eliminates global synchronization in every match-select-act cycle and takes advantage of a number of characteristics of production systems: the high ratio between reading and writing operations that allows the use of a simple bus as a communication network; the intensive need for searching that makes the use of associative memories effective; and the possibility of using a first solution matching algorithm that allows concurrency between matching and selecting.

The analytical model for the multiple functional unit Rete network addresses a shortcoming in the study of production system architectures: the rare use [7] of analytical modeling for performance evaluation. While, as expected, the model is not as accurate as those developed for more deterministic systems, it is a surprisingly effective and inexpensive tool for predicting system performance, trends, and for identifying where the bottlenecks are. In particular, it reveals that considerable amount of time is spent waiting in the queues of a standard implementation of a Rete Network. Thus a small number of functional units is sufficient to deliver considerable speed improvement in the match engine of a Production System machine. It also indicates that an appropriate configuration for the target application domain is formed by a moderate number of functional units in the Rete network and a modest number of processors. This reflects on the amount of parallelism available at two levels of granularity, for the production systems examined in this paper.

[7] To our knowledge, Yukawa *et al.* [43] made the only other attempt to use analytical modeling to estimate performace of the Rete Network

A Organization and Functioning of the Synchronizing Buffers

A.1 In-Order Buffer

Before the functioning of the IOB is described, some definitions and formalism are required. By convention, the children of a Rete network node are labeled as "left" and "right" children. All α-nodes are left children of the root node. The root node is the only parent of the α-nodes. Each β-node is associated with an antecedent of a production. A sign associated with the β-node indicates whether this antecedent is negated[8]. We will notate the sign of the i^{th} β-node with $\mathcal{S}(\beta_i)$. A β-node has two inputs (or parents). The left parent of a β-node is always the α-node that performs the α testing in the antecedent associated with the β-node. If the antecedent associated with a node β_i is negated, $\mathcal{S}(\beta_i) = -1$, otherwise $\mathcal{S}(\beta_i) = +1$.

The right parent of a β-node might be another β-node or an α-node. If it is an α-node, it must be the first antecedent of a production and therefore cannot be negated. If the right parent is a β-node, it is the result of a join operation of at least two independent antecedents of a production and cannot be negated either.

Two tokens are *conflicting* if one of them enables and the other disables the same production. Conflicting tokens can be identified locally at each β-node by the type of action specified by the tokens (addition or deletion of a WME) and the sign of the β-node to which they are destined.

The terminal nodes, also known as P-nodes store changes to the conflict set, These changes consist of addition or deletion of instantiation. Each production R_a has a corresponding P-node in the Rete network. A production that has a single antecedent has no need for join operations and no β-nodes. In this case the left son of an α-node is a P-node.

We say that a β-node β_i *belongs* to a production R_a, notated by $\beta_i \in R_a$ if there is a line of successors (or a path of β-nodes) in the Rete network that connects β_i to the P-node of R_a. In other words, if any token produced by β_i can eventually lead to the addition or removal of an instantiation

[8] A negated antecedent of a production tests for the absence of an specified WME in the working memory, while a non-negated antecedent tests for the presence of the specified WME. An instantiation of a production is created when all the non-negated antecedents are satisfied and there are no WMEs in the database that matches any of the negated antecedents.

of R_a, then $\beta_i \in R_a$. Notice that parts of the Rete network might be shared among productions, resulting in β-nodes that belong to more than one production.

The purpose of the In-Order Buffer is to prevent two conflicting tokens from proceeding to the β-units while allowing non-conflicting tokens to be processed without delay. At compile time, an analysis of the Rete network is performed by transversing the network from the α-nodes to the P-nodes following the left son links. This results in the construction of a two dimensional array with information about the relationship among the β-nodes. For each processor P_k, this array has two bits in each position and stores the function $\mathcal{C} : \mathcal{B} \times \mathcal{B} \rightarrow \{0, +1, -1\}$, where $\mathcal{B}$ is the set of all β-nodes in the Rete network.

$$\mathcal{C}(\beta_i, \beta_j) = \begin{cases} 0 & \text{if } \nexists R_a \in P_k / \beta_i \in R_a \text{ and } \beta_j \in R_a \\ +1 & \text{if } \exists R_a \in P_k / \beta_i \in R_a \text{ and } \beta_j \in R_a \text{ and } \mathcal{S}(\beta_i) = \mathcal{S}(\beta_j) \\ -1 & \text{if } \exists R_a \in P_k / \beta_i \in R_a \text{ and } \beta_j \in R_a \text{ and } \mathcal{S}(\beta_i) \neq \mathcal{S}(\beta_j) \end{cases}$$

The array formed with the values of $\mathcal{C}(\beta_i, \beta_j)$ is used to identify pairs of tokens that enable and disable the same production. The function of IOB is to prevent such tokens from being executed concurrently in the β-units. Observe that by definition $\mathcal{C}(\beta_i, \beta_i) = +1$ for any node β_i.

Tokens arriving at the Rete network are classified according to the action that they produce in the specified working memory element: add, delete, or modify. Because a modify token is pre-processed in the α-unit producing an add and a delete token, these are the only types of actions to be processed in β-nodes. We define the function $\mathcal{A} : \mathcal{T} \rightarrow \{+1, -1\}$, where $\mathcal{T}$ is the set of all possible tokens to be processed in any β-node.

$$\mathcal{A}(T_i) = \begin{cases} -1 & \text{if } T_i \text{ is a delete token} \\ +1 & \text{if } T_i \text{ is an add token} \end{cases}$$

After the processing in the α-nodes, a token is identified by a pair (T_i, β_i) that contains the token identification T_i and the identification of the β-node that will process the token. The decision of whether two tokens are conflicting takes into account both values. Suppose that a token (T_i, β_i) is already in IOB when a second token (T_j, β_j) arrives. The IOB decision to hold the second token until the processing of the first one is completed or to forward the new token immediately is based on the function $Conflict(i, j)$.

$$Conflict(i,j) = \mathcal{A}(T_i)\ \mathcal{A}(T_j)\ \mathcal{C}(\beta_i, \beta_j)\ +\ \varphi_{ij} \tag{28}$$

where

$$\varphi_{ij} = \begin{cases} -2 & \text{if } i = j \text{ and } S(\beta_i) = -1 \text{ and the right parent of } \beta_i \text{ is} \\ & \text{the first antecedent of a production} \\ 0 & otherwise \end{cases}$$

If $Conflict(i,j)$ is positive or equal to zero, there is no conflict between the two tokens and the second token to arrive in IOB can proceed to the external queue even if the processing of the first one has not been completed. Observe that if the tokens are destined to β-nodes that are not part of the same production, the value of $Conflict(i,j)$ is zero, independent of the types of action in the tokens. Two tokens destined to the same β-node are conflicting if their actions are different. A special situation occurs when the first β-node representing a production is negated. In this case both tokens with the same sign *and* tokens with opposite signs destined to that β-node might be conflicting[9]. When this situation occurs, φ_{ij} assumes value -2 and forces the value of $Conflict(i,j)$ to be negative independent of $A(T_i)$ and $A(T_j)$. For simplification of hardware, φ_{ij} can be stored as a forth value of $C(B_i, B_j)$, requiring no extra storage space.

Presence	T_i	β_i	$A(T_i)$	has_conflict	conflicts_with
⋮	⋮	⋮	⋮	⋮	⋮

Figure 12: Organization of In-Order Buffer (IOB)

The In-Order Buffer is an associative memory with the organization shown in Figure 12. It works as a queue with some additional decision logic. When a token arrives in the buffer, a single associative search identifies whether there is another token already in the buffer with which the new token conflicts. If no conflicting tokens are found in the buffer, the token is placed at the end

[9]To avoid this situation, some production system implementation do not allow the first antecedent of a production to be negated.

of the buffer with NULL value in its *conflicts_with* field. Simultaneously, the token is placed in the external queue to be processed by the β-units. If more than one conflicting token is found, lets assume that the one closer to the end of the buffer is stored in position k of the buffer. The newly arrived token is stored in the end of the buffer with the value k stored in its *conflicts_with* field. The single bit field *has_conflict* of the position k of the buffer is set to 1 indicating that the completion of the processing of the token in this position is been awaited by another token. Because there are no tokens waiting for the completion of the newly arrived token, its *has_conflict* field is initially reset.

IOB_Arrival(T_i, β_i,tail)

1 IOB(tail).$T_i \leftarrow T_i$
2 IOB(tail).$\beta_i \leftarrow \beta_i$
3 IOB(tail).*presence* $\leftarrow 1$
4 $k \leftarrow \max_j\{j \ / \ (T_j, \beta_j) \in$ IOB and $Conflict(k, j) < 0\}$
5 **if** $k = -1$
6 **then** IOB(tail).*conflicts_with* $\leftarrow$ NULL
7 **forward** (T_i, β_i) to β-units
8 **else** IOB(tail).*conflict_with* $\leftarrow k$
9 IOB(k).*has_conflict* $\leftarrow 1$
10 IOB(tail).has_conflict $\leftarrow 0$

In the algorithm that represents the sequence of steps for the arrival of a token in IOB, the argument tail is the first empty position at the end of the buffer, T_i is a unique id for the token arriving, and β_i is a unique id for the β-node that will process this token. IOB(tail).Ti and IOB(tail).β_i represent the T_i and the β_i fields of the first empty position at the end of the buffer, respectively. Step 4 is the associative search for possible conflicting tokens already in the buffer.

The processing of a token that is recorded in IOB is completed when the token and all its successors have been processed. This completion is signaled by the Output Buffer. When a token has been completely processed, if its *has_conflict* field is reset, the token is removed from IOB just by reseting the *presence* field. If the *has_conflict* field is set, an associative search in the *conflicts_with* field of the buffer identifies all tokens that were waiting for this completion. Suppose that one of

these tokens is found at position p of the buffer. The token at p might still conflict with some other token that arrived before the token now completed, but that has not been processed yet. Therefore, it is necessary to perform another associative search in the buffer to identify whether any other token ahead of p conflicts with p. If one such token is found at position q, the value of the *conflicts_with* field of p is changed to q, the field *has_conflict* of p is set, and the token in p remains waiting. If the token in p does not conflict with any other token ahead of itself, its *conflicts_with* field is set to NULL and the token is placed in the external queue.

Observe that the value stored in the field *conflicts_with* of a token that has many conflicting tokens in the IOB is always the one closest to the end of the buffer. It is likely that all other conflicting tokens will have left when this token leaves. Therefore the second search for further conflicting tokens will find none in most of the cases. Also, the presence of the single bit field *has_conflict* in the IOB prevents the execution of an associative search for tokens that do not have any conflicting token waiting for them.

IOB_Departure(T_i, β_i)

1 $k \leftarrow$ position of (T_i, β_i) in IOB
2 **if** IOB(k).*has_conflict* $= 1$
3 then **foreach** $p \in \{r$ / IOB(r).*conflicts_with* $= k\}$
4 **do** $q \leftarrow \max_j\{j$ / $(T_j, \beta_j) \in$ IOB and
$Conflict(p, j) < 0$ and $j \prec q\}$
5 **if** $q = -1$
6 **then** IOB(p).*conflicts_with* $\leftarrow$ NULL
7 **forward** (T_p, β_p) to β-units
8 **else** IOB(q).*has_conflict* $\leftarrow 1$
9 IOB(p).*conflicts_with* $\leftarrow q$

In the algorithmic presentation of the procedure for departure from IOB, steps 1, 3, and 4 involve associative searches. The relationship $j \prec q$ indicates that the token at position j precedes the token at position q in the buffer.

A.2 Output Buffer

The Output Buffer has the function of identifying the departure of a token from the system and signaling this departure to the IOB. Tokens processed in the Rete network are called "α-tokens" or "β-tokens". An α-token is a token that arrived at the α-unit. The processing of an α-token might produce a number of β-tokens to be placed in the external queue. Once a token arrives at the Rete network, it is assigned a unique tag. Every token generated internally also receives a unique identification. Whenever a β token is generated from an α-token, its tag is placed in the output-buffer.

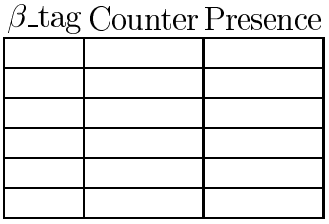

Figure 13: Organization of Output Buffer (OB)

The execution of a β-token might produce further tokens. We call these tokens generated at the β-units *internal tokens*. For all practical purposes there is no distinction between a β-token and an internal token other than that the former is produced in an α-unit while the later is produced in a β-unit. The identification of a β-token is carried along by all its successors. Whenever a new internal token is produced, the counter corresponding to its original β-token is incremented in the OB. When an internal token is processed, this counter is decremented. Whenever a counter reaches zero, the β-token identification is removed from the buffer and the IOB is notified of the token departure.

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] J. N. Amaral and J. Ghosh. Versatile benchmarking for concurrent production system architectures. In *XV Congress of the Brazilian Computer Society*, pages 599–610, July 1995.

[7] J. N. Amaral and J. Ghosh. A concurrent architecture for serializable production systems. *IEEE Transactions on Parallel and Distributed Processing*, 7(12):1265–1280, December 1996.

[8] F. Barachini and N. Theuretzbacher. The challenge of real-time process control for production systems. In *Proceedings of National Conference on Artificial Intelligence*, pages 705–709, August 1988.

[9] R. Bechtel and M. C. Rowe. Parallel inference performance prediction. In *Proceedings of the IEEE 1990 National Aerospace and Electronics Conference - NAECON*, pages 21–25, May 1990.

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

[11] H. G. Cragon. *Memory Systems and Pipelined Processors.* Jones and Bartlet Publishers, Sudbury, MA, 1996.

[12] R. B. Doorenbos. *Production Matching for Large Learning Systems.* PhD thesis, Carnegie Mellon University, January 1995. CMU-CS-95-113.

[13] R. D. Doorenbos. Matching 100,000 learned rules. In *Proc. 11th National Conference on Artificial Intelligence*, pages 290–296, 1993.

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

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

[16] J.-L. Gaudiot and A. Sohn. Data-driven parallel production systems. *IEEE Transactions on Software Engineering*, 16:281–291, March 1990.

[17] A. Gupta. Implementing OPS5 production systems on DADO. In *Proceedings of International Conference on Parallel Processing*, pages 83–91, 1984.

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

[19] A. Gupta, M. Tambe, D. Kalp, C. L. Forgy, and A. Newell. Parallel implementation of OPS5 on the Encore multiprocessor: Results and analysis. *International Journal of Parallel Programing*, 17, 1988.

[20] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability and Programmability.* McGraw-Hill, New York, 1993.

[21] T. Ishida. An optimization algorithm for production systems. *IEEE Transaction on Knowledge and Data Engineering*, 6(4):549–558, August 1994.

[22] 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.

[23] T. Ishida, M. Yokoo, and L. Gasser. An organizational approach to adaptive production systems. In *Proceedings of National Conference on Artificial Intelligence*, pages 52–58, July 1990.

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

[25] W. M. Johnson. *Superscalar Microprocessor Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[26] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, New York, 1993.

[27] M. A. Kelly and R. E. Seviora. An evaluation of DRete on CUPID for OPS5 matching. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 84–90, August 1989.

[28] M. A. Kelly and R. E. Seviora. A multiprocessor architecture for production system matching. In *Proceedings of National Conference on Artificial Intelligence*, pages 36–41, July 1989.

[29] H. Kikuchi, T. Yukawa, K. Matsuzawa, and T. Ishikawa. Presto: A bus-connected multiprocessor for a Rete-based production system. In *Joint International Conference on Vector and Parallel Processing - CONPAR 90*, pages 63–74, February 1990.

[30] L. Kleinrock. *Queueing Systems Volume I: Theory*. John Wiley & Sons, New York, 1975.

[31] 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.

[32] H. S. Lee and M. I. Schor. Match algorithms for generalized Rete Networks. *Artificial Intelligence*, 54:249–274, April 1992.

[33] J. D. C. Little. A proof of the queueing formula $l = \lambda w$. *Operations Research*, 9:383–387, 1961.

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

[35] M. C. Rowe, J. Labhart, R. Bechtel, S. Matney, and S. Carrow. Forward chaining parallel inference. In *Proceedings of the Second IEEE Symposium on Parallel and Distributed Processing*, pages 455–462, December 1990.

[36] J. G. Schmolze and S. Goel. A parallel asynchronous distributed production systems. In *Proceedings of National Conference on Artificial Intelligence*, pages 65–71, July 1990.

[37] F. Schreiner and G. Zimmermann. PESA I - a parallel architecture for production systems. In *Proc. 1986 International Conference of Parallel Processing*, 1986.

[38] A. Sohn and J.-L. Gaudiot. A macro actor/token implementation of production systems on a data-flow multiprocessor. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 36–41, August 1991.

[39] S. J. Stolfo, D. Miranker, and D. E. Shaw. Architecture and applications of DADO: A large-scale parallel computer for artificial intelligence. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 850–854, August 1983.

[40] S. J. Stolfo and D. P. Miranker. DADO: A parallel processor for expert systems. In *Proceedings of International Conference on Parallel Processing*, pages 74–82, April 1984.

[41] J.-H. Wang, J. Srivastava, and W. T. Tsai. A transaction model for parallel production systems: Part ii. model and evaluation. *International Journal on Artificial Intelligence Tools*, 2(3):431–457, 1993.

[42] J. A. White, J. W. Schmidt, and G. K. Bennett. *Analysis of Queueing Systems*. Academic Press, New York, 1975.

[43] T. Yukawa, T. Ishikawa, H. Kikuchi, and K. Matsuzawa. TWIN: A parallel scheme for a production system featuring both control and data parallelism. In *Proceedings of the 7th Conference on Artificial Intelligence Applications*, pages 64–70, February 1991.

Biographies

José Nelson Amaral received his B.S. in Electrical Engineering from Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS-Brazil) in 1987. He received his M.E. from Instituto Tecnológico de Aeronáutica (Brazil) in 1989 and his Ph.D. in Computer Engineering from The University of Texas at Austin in 1994. He is currently a faculty member and the Graduate Coordinator in the Electrical Engineering Graduate Program at PUCRS. He received fellowships from the Brazilian government to pursue his graduate studies. Besides his work with innovative distributed symbolic architectures, his current research interests include the solution of complex engineering problems through biological inspired solutions such as Artificial Neural Networks, Genetic Algorithms, and Asynchronous Teams.

Joydeep Ghosh completed his B.Tech from IIT Kanpur in 1983, and MS and Ph.D. in Computer Engineering from the University of Southern California, in 1985 and 1988 respectively. He was the first member of the School of Engineering at USC to be awarded an "All-University Predoctoral Teaching Fellowship" for four years.

Dr. Ghosh is currently an Associate Professor in the Department of Electrical and Computer Engineering at the University of Texas, Austin. His research interest lie in the areas of intelligent and adaptive systems, including artificial neural networks and knowledge based systems, and he has published over 100 refereed papers on these topics. Dr. Ghosh received the 1992 Darlington Award given by the IEEE Circuits and Systems Society for Best Journal Paper and also "best conference paper" citations for four neural network theory or application papers. He served as the general chairman for the SPIE/SPSE Conference on Image Processing Architectures, Santa Clara, Feb. 1990, and as Conference Co-Chair of Artificial Neural Networks in Engineering (ANNIE)'93 ANNIE'94, and ANNIE'95, and in the program committee of several conferences on neural networks and parallel processing. He is a member of the editorial board of *Pattern Recognition*, *IEEE Trans. on Neural Networks* and *Neural Computing Surveys*.