TWIN: A Parallel Scheme for a Production System
Featuring both Control and Data Parallelism

Takashi Yukawa, Tsutomu Ishikawa, Hideo Kikuchi and Kazunitsu Matsuzawa
NTT Communications and Information Processing Laboratories
yukawa@nttaip.ntt.jp@relay.cs.net

Abstract
This paper proposes a new parallel scheme, called TWIN, for Rete-based production systems. The TWIN scheme exploits two aspects of the Rete algorithm parallelism: control parallelism in the structural features of the Rete network and data parallelism in token comparison at two-input nodes. TWIN is suited for building a simple, compact, bus-connected parallel system without shared memory or cache memory. An analytical model of the Rete network is presented which estimates the increase in speed and makes it possible to evaluate the TWIN scheme applied to a wide spectrum of production system parameters. Experimental results are also shown to demonstrate the performance of this scheme and to verify of the analytical model.

A1 topic: Production systems, pattern matching
Domain area: Rete pattern matching
Language/Tool: C language
Status: Implemented as experimental system
Impact: Attains estimated 16-fold speed increase on a 16PE simple bus-connected multi-processor; superior to other schemes on this architecture.

1 Introduction
Recently, OPS5-like production systems[1] have been extensively employed in building larger and more complex expert systems, especially in industry. However, these production systems are often slow in execution, which limits their usefulness. To increase execution speed, various parallel architectures have been proposed that focus on the matching phase[2, 3, 4, 5, 6], which accounts for more than 90% of the execution time. This paper also focuses on parallelism in matching phase.

An efficient match algorithm called Rete was proposed by Forgy[7] and is commonly used for OPS5-like production systems. Parallel architectures that exploit the fine-grained parallelism of the Rete algorithm have been the most widely proposed[3, 5]. Although exploiting finer granularity provides higher parallelism and a better load balance, this increases both the amount of data transfer between processors and the number of synchronizing requirements. The broad data transfer bandwidth and cache memory sub-system, which reduce communication and synchronization overheads, make the hardware complicated and expensive.

By exploiting coarse-grained parallelism, overhead can be reduced, but this generally leads to load imbalance because of the differences in processing time required for different parallel processes. In a practical production system applications, there are few active grains because only a few productions are affected by each rule firing. This fact shows that the speed loss due to load imbalance becomes a serious problem for most programs.

This paper presents a new scheme called TWIN (from top two-input node) based on coarse-grained parallelism. This scheme provides fewer requirements for data transfer and synchronization by exploiting the separative features of the Rete network, and provides a good load balance by exploiting data comparison parallelism in the Rete network nodes. It is suited for building a simple, compact, bus-connected parallel system without shared memory or cache memory.

This paper also presents a new evaluation method for parallel schemes. Most papers[3, 4] have used simulation results to evaluate the performance of parallel schemes. Our evaluation differs from these in that an analytical approach is adopted to estimate production system performance for a wide range of parameters. For this evaluation, we propose an analytical model of the Rete network and then express the speed increase equation using production system program parameters based on the Rete network model. The analytical model provides estimates which are rough but sufficiently precise to compare the performance of various parallel schemes for a wide range of production system parameters without running an actual program.

Finally, this paper describes the experimental results in order to demonstrate the performance of the TWIN scheme and to validate of the analytical model. For the experiment, we parallelize and implement a C-language interpreter program on a bus-connected multi-processor and use a routing problem solver as a sample expert system.

The paper is organized as follows. Section 2 gives background information on production systems and previous work on improving their performance. Section 3 describes in detail a parallelizing strategy for the TWIN scheme; section 4 gives analytical estimates of speed increases obtained by statistical modeling of the Rete network. Section 5 presents experimental results and compares these with the analytical results. Section 6 summarizes the results and presents conclusions.
2 Background

This section presents an overview of production systems and previous work on their performance.

2.1 Production System

A production system consists of an unordered collection of IF – THEN rules called productions and a global database of temporary assertions called working memory. The assertions of the working memory are called working memory elements (WMEs). By convention, the IF part of a production is called its LHS (left-hand side) and the THEN part is called the RHS (right-hand side).

The production system interpreter executes a three-phase operation called the recognize-act cycle repeatedly. The three phases in the cycle are:

- **Match**: In this phase, all of the productions are matched against the current contents of the working memory to find which productions have their LHS satisfied. A set of a production and WMEs that satisfies its LHS, called instantiation, is stored into a conflict set.

- **Conflict-Resolution**: In this phase, an instantiation is selected from the conflict set for execution. If the conflict set is empty, the interpreter halts.

- **Act**: In this phase, the RHS of a production selected in the above phase is executed. In general, the RHS of a production describes creation, modification and deletion of WMEs.

2.2 Rete Match Algorithm

Most of the execution time of a production system is consumed in the match phase of the recognize-act cycle. The Rete algorithm[7] stores state information between recognize-act cycles, making it efficient for the match phase. The productions are compiled into a kind of data flow network called a Rete network, and descriptions of working memory changes, called tokens, are propagated along the arcs connecting the nodes in the network.

Figure 1 shows an OPS5-like left associated Rete network. The nodes in the Rete network include a root node, constant-test nodes, two-input nodes, terminal nodes, and memory nodes. The constant-test nodes test single constant attribute values within a condition element. The two-input nodes check the consistency of variable bindings between two condition elements. If a token reaches a terminal node, it is stored in the conflict set as an instantiation. The memory nodes store tokens that have passed the constant-test nodes and two-input nodes to avoid performing the same tests for each cycle. The right input of a two-input node always comes from a constant-test node, while its left input can come from a constant-test node or a two-input node. Identical nodes required by more than two productions are shared rather duplicated. Storing the partial results of the match as states within memory nodes and sharing identical nodes make the Rete algorithm efficient.

2.3 Evaluation of Conventional Scheme

There are two aspects of parallelism in the Rete algorithm. One is control parallelism based on a structural feature of the Rete network associated with, for example, each production or each node. The other is data parallelism, which is simultaneous comparison of tokens at two-input nodes. When a token arrives at an input of a two-input node, the token is compared with all tokens stored in the memory node connected with the opposite input of the two-input node. Comparisons in the two-input node are independent of each other, so they can be processed in parallel.

Since parallelism in a production system changes dynamically and has less locality, load imbalance and communication overhead limit the speed increase. To obtain greater speed increase, various parallel schemes have been proposed and implemented over the past few years.

The production-level scheme which divides the network into coarse-grained processes based on an individual production or a set of productions has the advantage of requiring no communication between the match processes. However, only a small increase in speed is expected because a lower average number of rules is affected by changes in working memory and there is a large variation of processing time in each production[2].

The node-level scheme which divides the Rete network into processes based on an individual node provides finer granularity. However, this leads to increased communication requirements between processes evaluating the nodes in parallel. Gupta at Stanford University implements the node-level scheme on a bus-connected multi-processor[8]. This implementation requires shared memory, a high performance bus and high-speed cache memory subsystems to reduce communication overhead.

Kelly at the University of Waterloo proposes a scheme which exploits the token comparison-level parallelism with node-level parallelism[5]. This scheme has the finest granularity of the three, therefore it provides less variance in the processing time than the node-level scheme or the production-level scheme. However, it requires more communication between processes. The high performance inter-
processor connections and specially designed processors are required to make full use of the potential of this scheme.

A scheme combining the token comparison-level parallelism with the production-level parallelism has not been proposed because the token comparison-level parallelism has been considered to spoil the advantage of production-level parallelism due to increasing communication requirement.

3 The TWIN scheme

The TWIN scheme exploits the token comparison-level parallelism with the production-level parallelism without communication overhead. This scheme targets a simple bus-connected parallel processor system with 10-20 processors such as microprocessors connected by a universal bus [9]. TWIN requires no shared memory because it divides the Rete network into highly independent tasks.

3.1 Parallelizing Strategy

The TWIN scheme, which is applicable to an OPS5-like left associated Rete network, is based on production-level parallelism. It achieves good load balance without increasing communication requirements by exploiting coarse-grained structural parallelism combined with token comparison parallelism.

The details of TWIN are as follows.

(1) Structural Parallelizing

In the TWIN scheme, the Rete network is divided into several subnetworks based on the top two-input nodes, as shown in Figure 2. Each subnetwork consists of the top two-input node (TW1 or TW3 in Figure 2), all successive two-input nodes, and all preceding constant-test nodes connected to the two-input nodes. Subnetworks operate in parallel because there is no interaction between them. Constant-test nodes connected to several two-input nodes are copied into each subnetwork. This duplication of constant-test nodes does not cause a significant loss in speed because the constant-test nodes require less processing time than the two-input nodes. This scheme has the advantage that no two-input nodes are copied into subnetworks, as is the case with the conventional production-level scheme having similar granularity.

(2) Token Comparison Parallelizing

To exploit token comparison parallelism, a subnetwork is copied and then each copy forms a task. When a token created by rule firing is sent to all tasks, each copy starts processing its designated nodes without any interaction with other tasks. The match is executed in the same way as the conventional coarse-grained scheme, except that only one task stores the received token in its memory node. No other task ever stores this token. Figure 3 illustrates how the task examines the received token in the subnetworks. The token flow is as follows.

1. Each task processes the constant-test node (CT1_A or CT1_B in Figure 3). If matched, the token passes through to the successor node; otherwise the tasks quit the match phase.
2. If the successor node is the top two-input node (TW1_A or TW1_B),
   - Only one of the tasks stores the token in the memory node (M1_B in subnetwork B).
   - Each task compares the token with the tokens in the opposite memory node (M2_A or M2_B).
3. If the successor node is a non-top two-input node (not shown in Figure 3),
   - Each task stores the token in the memory node.
   - Each task compares the token with tokens in the opposite memory node.
4. Tasks repeat step 3 until a terminal node appears.

The TWIN scheme has the following features.

- Each task can independently determine whether it should store the token if the token has an identification number. If the identification numbers are assigned randomly, tokens stored in memory nodes will be equally distributed among the tasks after many cycles.
- Nodes under the top two-input nodes are driven by different tokens generated at the top two-input nodes. The tokens are independent of each other so they can be processed in parallel without communication.
- The deletion of tokens, which is caused by removing WMEs, is also executed in parallel because tokens that should be removed from memory nodes are equally distributed.

Consequently, the TWIN scheme achieves good load balance without communication overhead, and a larger increase in speed is expected even for small programs.

3.2 Mapping of Tasks

In TWIN, tasks are statically assigned before starting interpretation to avoid task migration overhead. To provide uniform PE load, all PEs must have the same number of activated nodes. This can be achieved by equalizing the number of activated constant-test nodes in PE because an incoming token activates the two-input nodes via a constant-test node which has the same class as the token.
Assume that a production system program contains $n$ classes ($C_1, C_2, ..., C_n$), and that there are $m$ PEs ($PE_1, PE_2, ..., PE_m$). The variance of the number of constant-test nodes which belong to a class $C_i$ is given as:

$$V_i = \frac{1}{m} \sum_{j=1}^{m} (N_{ij} - \frac{1}{m} \sum_{k=1}^{m} N_{ik})^2,$$

where $N_{ij}$ is the number of constant-test nodes that belong to the class $C_i$ and are included in the $PE_j$.

The subnetworks are assigned to PEs such that the sum of $V_i$s is minimized.

### 4 Analytical Speed-up Estimation

In this section, we discuss the analytically estimated increase in speed of the TWIN scheme. First, a Rete network model which focuses on statistical behavior is described. Then, the speed increase estimates of the TWIN scheme obtained using the model are presented. Finally, the analytical results are shown and performance improvement due to structural parallelism and token comparison parallelism is analyzed.

#### 4.1 Rete Network Model

The features of the Rete network are characterized by two-input nodes because most of the processing time in the match phase is spent on them. In addition, all other kinds of nodes in the Rete network are connected to the two-input nodes. If we construct a two-input node model, the features of the Rete network can be expressed using the surface parameters shown in Table I, which can easily be obtained by program surface measurements.

The two-input node that this model is based on has the following features, which were obtained from observation of a real production system program[10]:

- The two-input node has a few branch outputs that occur only when the node is shared between productions.
- The number of productions that have a certain number of two-input nodes decreases exponentially with increasing the number of two-input nodes.

The two-input node model can be expressed as in Figure. 4 according to these features. Statistically the model has $B$ outputs, one of which is connected to the terminal node with constant probability $P$. Thus, the frequency of the length of the two-input node chain decreases exponentially[11] and has an average value of $e$, given by

$$e = \frac{1}{1 - (B - P)}.$$

The distribution of the number of condition elements per production has the same shape as the distribution of the number of two-input nodes per production because the features of the Rete network are characterized by two-input nodes because most of the processing time in the match phase is spent on them. In addition, all other kinds of nodes in the Rete network are connected to the two-input nodes. If we construct a two-input node model, the features of the Rete network can be expressed using the surface parameters shown in Table I, which can easily be obtained by program surface measurements.

<table>
<thead>
<tr>
<th>Table I: Surface parameters of a production system program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of productions $R$</td>
</tr>
<tr>
<td>Number of condition elements per production $E$</td>
</tr>
<tr>
<td>Number of action elements per production $A$</td>
</tr>
<tr>
<td>Number of working memory elements $W$</td>
</tr>
<tr>
<td>Number of classes $C$</td>
</tr>
<tr>
<td>Number of distinct condition elements $N$</td>
</tr>
<tr>
<td>Probability of success in a constant-test node $\alpha_c$</td>
</tr>
<tr>
<td>Number of output branches on two-input node $B$</td>
</tr>
<tr>
<td>Probability of connection to terminal node $P$</td>
</tr>
</tbody>
</table>
Table II: Features of the Rete network

<table>
<thead>
<tr>
<th>Number of constant-test node</th>
<th>$D_1 = N_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant-test node sharing rate</td>
<td>$S_c = R(E - 1)/D_1$</td>
</tr>
<tr>
<td>Number of two-input nodes</td>
<td>$D_2 = R/P$</td>
</tr>
<tr>
<td>Connections between constant-test nodes and two-input nodes</td>
<td>$S = D_2(2 - (B - P))$</td>
</tr>
<tr>
<td>Constant-test nodes activated by a token</td>
<td>$O_t = D_1/C$</td>
</tr>
<tr>
<td>Tokens that pass constant-test nodes</td>
<td>$X_t = S(C/E)$</td>
</tr>
</tbody>
</table>

The number of two-input nodes is equal to the number of condition elements minus one. Hence, the relation between $B$, $P$ and the number of condition elements per production, $E$, is

$$P = (B - 1) + \frac{1}{E - 1}.$$ 

The number of condition elements, $E$, for a given production system program is obtained by surface measurement. The branch $B$ is normally in the range of 1.1 - 1.3, and may go as high as 1.7, even for a program that has many shared condition elements[10].

The Rete network is constructed of two-input node chains. Therefore, the features of the Rete network are formulated by the properties of the two-input node model, which is expressed by program surface parameters. Table II shows the structural features of the Rete network[11].

4.2 Estimation of Speed Increase for the TWIN scheme

In the TWIN scheme, there is no communication overhead. However, the following factors limit the increase in speed.

- Dividing a problem leads to additional operations because some part of the problem cannot be localized in a PE. Let $Q$ be the overhead coefficient due to problem dividing.
- The overall execution time is limited by $t_{\text{max}} = \max\{t_1, t_2, \ldots\}$, where $t_i$ is the processing time for $PE_i$. Let $t_{\text{ave}}$ be the average of $t_1, t_2, \ldots$ and let $\gamma$ be $t_{\text{max}}/t_{\text{ave}}$.

The execution time, $T$, for parallel processes is:

$$T = \frac{Qt_{\text{ave}}}{N},$$

where $T_{\text{ave}}$ is the processing time for sequential processing, and $N$ is the number of PEs. Hence, the increase in speed $U$ is given by

$$U = \frac{T_{\text{ave}}}{T} = \frac{N}{Q} \gamma.$$

The parameters $Q$ and $\gamma$ can be estimated on the basis of the Rete network model. How the parameters determined is demonstrated below.

$Q$: The overhead coefficient of additional operation
In TWIN, constant-test nodes which belong to more than two subnetworks are copied and processed in different PEs. Moreover, constant-test nodes included in duplicated subnetworks to exploit token comparison parallelism are also processed in different PEs. Consider that a subnetwork is duplicated into $M_d$ tasks to exploit token comparison parallelism. The number of constant-test nodes performed on the entire PEs is $S_d M_d$ times as many as the original number of constant-test nodes. Let $R_c$ be the ratio of the execution time consumed on constant-test nodes to the overall matching time. The coefficient of additional operation is given as

$$Q = (1 - R_c) + S_d M_d R_c.$$ 

$\gamma$: Load deviation ratio ($t_{\text{max}}/t_{\text{ave}}$)
Both $t_{\text{max}}$ and $t_{\text{ave}}$ can be determined from the probability distribution of the execution time, $P(t)$, the number of PEs, $N$, and the number of activated tasks assigned to each PE, $K$.

In the TWIN scheme, since $P(t)$ has exponential distribution, we can compute the probability distribution $P'(T)$, where $T$ is the processing time for $K$ tasks. The average of processing time, $t_{\text{ave}}$, is just the average of $P'(T)$, $P'(T)$, and it is equal to $K P(t)$. We can also pick up $N$ samples $T_1, T_2, \ldots, T_N$ randomly out of a set that has probability distribution $P'(T)$ and find the maximum sample, $T_{\text{max}}$. Iterating above procedure, $t_{\text{max}}$ is estimated as $\gamma T_{\text{max}}$. Therefore, $\gamma$ results in

$$\gamma = \frac{T_{\text{max}}}{K P(t)}.$$ 

In the TWIN scheme, the number of activated subnetworks is equal to the number of tokens $X_1$ that pass the constant-test nodes. Therefore, $K$ is given by:

$$K = \frac{X_1}{N} M_d.$$ 

We can prepare a table of $\gamma$ for various $N$ and $K$.

4.3 Analytical Results

The performance of TWIN is estimated for a production system according to the analysis mentioned above.

A speed increase of the TWIN scheme is shown in Figure 5. The production system parameters used for the evaluation are shown in Table III. The ratio of the execution
Table III: Program parameters for analytical evaluation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>100</td>
<td>$C$</td>
<td>5</td>
</tr>
<tr>
<td>$E$</td>
<td>3.5</td>
<td>$B$</td>
<td>1.2</td>
</tr>
<tr>
<td>$A$</td>
<td>2.5</td>
<td>$N_i$</td>
<td>$2 \times R$</td>
</tr>
<tr>
<td>$W$</td>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time consumed on constant-test nodes to the whole matching time, $R_c$, depends on implementation and is assumed to be 5%. The selection of these parameters is reasonable since they often appear in actual programs. In Figure 5, if bus-connected parallel processor are adopted, the TWIN scheme can be seen to be superior to other finer grained schemes. A speed increase of the node-level scheme, which seems suitable for a bus-connected processor, is shown by dotted line for this comparison. The speed increase of the node-level scheme can be estimated in a way similar to TWIN. The token transfer cost is assumed to be equal to the token pair comparison cost in a two-input node.

Figure 6 shows the number of productions versus the speed increase of the TWIN scheme on 16 PEs. We used the same parameters mentioned above except for the number of productions. The dotted line shows the speed increase of production parallelism with no token comparison and the solid lines show the speed increase with $M_d = 3$ and with $M_d = 5$. The graph demonstrates that exploiting token comparison parallelism improves the speed increase for the whole range of productions and that it improves the speed increase significantly for fewer productions.

Figure 7 shows the increase in speed of the match phase for the TWIN scheme with and without exploiting data comparison parallelism. The solid lines show experimental results and the dotted lines show analytical results.

In Figure 7, the TWIN scheme without token comparison

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>20</td>
<td>$C$</td>
<td>5</td>
</tr>
<tr>
<td>$E$</td>
<td>2.7</td>
<td>$B$</td>
<td>1.0</td>
</tr>
<tr>
<td>$A$</td>
<td>1.0</td>
<td>$N_i$</td>
<td>57</td>
</tr>
<tr>
<td>$W$</td>
<td>700</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5 Experimental Results

In this section, we describe results obtained from the execution of a real production system program and demonstrate the performance of the TWIN scheme and the correctness of the analytical model.

For the experiment, we parallelized and implemented the KBMS/C rule interpreter, an OPS5-like production system developed by NTT, on the bus-connected multi-processor. Each PE is comprised of a general-purpose microprocessor and local memory, and is connected to a common bus which is capable of both point-to-point data transfer and broadcast data transfer. The sample program is a routing problem solver which has the surface parameters shown in Table IV. The program finds the minimum cost route from one point to another point on the basis of an ordered depth-first search algorithm.

Figure 8 shows the increase in speed of the match phase for the TWIN scheme with and without exploiting data comparison parallelism. The solid lines show experimental results and the dotted lines show analytical results.

In Figure 8, the TWIN scheme without token comparison

![Figure 5: Estimated Speed Increase of the TWIN scheme](image)

![Figure 6: Speed Increase versus the Number of Productions](image)
parallelism can achieve only a 2-fold speed increase. The reason for this small speed increase is load imbalance. Since the sample program has only 20 rules, the fact that there are only a few rules affected in each recognize-act cycle results in a large processing time variance in each PE. On the other hand, the TWIN scheme with token comparison parallelism can achieve a fairly good speed increase. Since the token comparison parallelism increases the number of active tasks by processing the same subnetwork in parallel, load balance is improved and higher speed increase is achieved.

The graph also demonstrates that the analytical model provides a fairly good estimate of the speed increase. The differences between analytical and experimental results are 10% at most. This is sufficiently precise to characterize the speed increase of the parallel scheme without running a actual program.

6 Conclusion

This paper has presented a new parallel scheme called TWIN for production systems. TWIN achieves good load balance in the matching phase without communication requirements by exploiting coarse-grained structural parallelism combined with token comparison parallelism. This scheme is suited for building a simple, compact bus-connected parallel system without shared memory or cache memory.

An analytical evaluation of the TWIN scheme based on a Rete network model is also described. The model can estimate the speed increase in various production system programs without actual execution. The analytical results show that TWIN is superior to the other finer grained schemes when simple bus-connected processor is adopted.

To demonstrate the performance of the TWIN scheme and the validity of the analytical model, an experimental measurement using a bus-connected multi-processor is presented. The experimental results demonstrate that exploiting token comparison parallelism improves performance even for a small production system program. Higher speed increase is expected for larger programs.

In addition, this result proves that the analytical model provides a fairly good estimate of speed increase. The analysis is useful in evaluating various parallel schemes for a wide range of production system parameters.

Acknowledgments

The authors wish to thank Akio Mizugaki for his many contributions to the experimental system development.

References