CHAPTER 5

PARALLEL EVALUATION OF CENSORED PRODUCTION RULES
A Censored Production Rule is a standard production rule augmented with the UNLESS operator. The general form of the CPR can be represented as:

\[
\text{IF } P_1 \text{ and } P_2 \text{ and } \ldots \text{ and } P_k \\
\text{THEN } D \\
\text{UNLESS } C_1 \text{ or } C_2 \text{ or } \ldots \text{ or } C_l
\]

The disjunction operator between the censors indicates that the entire decision would be reversed as soon as any one of them evaluates to TRUE. In sequential evaluation, the censors are evaluated starting from left. The failure of the censor reconfirms the decision, whereas the truth of the censor negates the decision. If one of the censor towards the end of the censor list evaluates to true, then the entire effort in evaluating the earlier censors to reaffirm the rule is wasted. Thus it would be worthwhile to evaluate them in parallel. The disjunction property of the censors ensure that they can be evaluated in parallel with minimum synchronization overheads. Further as all censors of the same parent are independent of each other they can be evaluated asynchronously. In the subsequent sections we have exploited these properties to propose the parallel implementation of censors.

5.1 The Sequential Evaluation

Each of the premises and censors of a rule, of the type given above, can again be rules with their own set of premises and censors. Thus, the entire rule can be represented graphically in the form of a tree as shown in Fig 5.1, where $P_i$ represents a premise and $C_j$ a censor. $P_{n(p(i))p(i)}$ represents the $n(p(i))$th premise of the premise $P_i$ while $C_{m(p(j))c(j)}$ the $m(p(j))$th censor of the censor $C_j$ and so on.
Figure 5-1: A general Censored Production Rule tree
The rule is evaluated depth by depth starting from censor chaining depth 0. Thus if there is not enough time available, the rule can be evaluated only upto 0th censor chaining depth, implying, evaluating only the premises of the rule. The rules of fig 5.1 would thus look like

\[\text{IF } P_1 \text{ and } P_2 \ldots \text{ and } P_i \]
\[\text{THEN } D\]

\[\text{IF } P_1 p(i) \text{ and } P_2 p(i) \text{ and } \ldots \text{ and } P_n(p(i)) p(i) \]
\[\text{THEN } P_i\]

etc. The process of evaluation of premises continues until all the premises have been evaluated completely, or any of the premise evaluates to FALSE.

As the next step, the rule is evaluated upto censor chaining depth 1. This would mean evaluating only the censors directly connected to the rule, or the censors which have only premises as their ancestors. For fig 5.1, the rules would thus look like

\[\text{IF } P_1 \text{ and } P_2 \text{ and } \ldots \text{ and } P_i \]
\[\text{THEN } D\]
\[\text{UNLESS } C_1 \text{ or } C_2 \text{ or } \ldots \text{ or } C_j\]

\[\text{IF } P_1 p(i) \text{ and } P_2 p(i) \text{ and } \ldots \text{ and } P_n(p(i)) p(i) \]
\[\text{THEN } P_i\]
\[\text{UNLESS } C_1 p(i) \text{ or } C_2 p(i) \text{ or } \ldots \text{ or } C_m(p(i)) p(i)\]

\[\text{IF } P_1 c(j) \text{ and } P_2 c(j) \text{ and } \ldots \text{ and } P_n(p(j)) c(j) \]
\[\text{THEN } C_j\]

The censors of the censors are not evaluated for depth 1. For depth 2, the above rule takes the form

\[\text{IF } P_1 c(j) \text{ and } P_2 c(j) \text{ and } \ldots \text{ and } P_n(p(j)) c(j) \]
\[\text{THEN } C_j\]
\[\text{UNLESS } C_1 c(j) \text{ or } C_2 c(j) \text{ or } \ldots \text{ or } C_m(c(j)) c(j)\]

To understand the process of evaluation of rules, let us consider the example of CPRs discussed in section 3.1 and subsequently look at fig 3.1 to get a
graphical representation of the rules set. The evaluation process starts by evaluating only the premises of the rule. The censors encountered in the process are saved for later use. For fig 3.1, it would imply evaluating $P_1$, $P_2$ and $P_4$ and saving $C_1$, $C_2$ and $C_4$ for later use.

The process of evaluation for censor chaining depth 0 continues until all the rules associated with that depth have been fully evaluated, or any premise has evaluated to FALSE. In the latter case, the result of the premise is simply propagated back, the overall decision of the rule D is reversed and the evaluation process terminated. On the contrary, if the result of all the premises up to depth 0 is TRUE, the decision is said to generally hold. Censor chaining depth 1 containing censors from depth 0 are evaluated only if there is more time available with the user generating the query. If time permits, the censors $C_1$, $C_2$ and $C_4$ are restored and the process of evaluation is resumed. The censors are said to be at depth 1. Evaluation of $C_1$, $C_2$ and $C_4$ proceeds by as follows:

<table>
<thead>
<tr>
<th>Censor restored</th>
<th>Premises to be evaluated</th>
<th>Censor Saved</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>$P_3, P_6$</td>
<td>$C_3, C_6$</td>
</tr>
<tr>
<td>C2</td>
<td>$P_5$</td>
<td>$C_5$</td>
</tr>
<tr>
<td>C4</td>
<td>fact</td>
<td>none</td>
</tr>
</tbody>
</table>

The results in each case are accumulated and evaluated to arrive at the final decision. The evaluation process continues until:

- the complete rule has been evaluated
- the time limit permits
- decision with required certainty is reached

In case of time limit, the rule is evaluated level by level starting from the 0th censor chaining depth, until the time limit set by the user expires. After each level of evaluation, the time elapsed is compared with the time limit available.
The evaluation continues, only if there is more time available. At the end of the evaluation, the certainty factor database of the rule is looked upon to get the certainty in the decision at that censor chaining depth.

Instead of giving time as the limit, the user may desire to have the decision with a given certainty. In such a condition, the certainty factor database, containing certainty factors at various depths for the rule, is scanned, to see upto what censor chaining depth, the rule needs to be evaluated to arrive at the decision with the desired certainty. The process of evaluation then continues until the censors have been evaluated upto the required censor chaining depth. The results at the end of the evaluation process are combined to get the final value of the corresponding censor. In the process, the decision could have achieved that much of certainty or the polarity of the decision could have been reversed, depending upon the knowledge base.

For both the above cases, the rule is evaluated, and the results of each of the premises and censors are tabulated at each censor chaining depth, and subsequently propagated back at the end of the evaluation process to the main root. Thus the total evaluation process of the rule can be divided into two phases:

1. the evaluation phase - in which the premises and the censors of the rule are evaluated.
2. the propagation phase - in which the results are combined and propagated back to the main root.

After the complete rule has been evaluated, the decision is either FALSE, or TRUE with a given certainty. The certainty in the decision increases with the increased censor chaining depth and is set by the knowledge engineer in the form of a database with each rule.
5.2 The Parallel Implementation

Looking at the sequential evaluation of censors, we find there are two phases involved in the evaluation of CPRs. These are the evaluation phase and the propagation phase. The evaluation phase involves evaluating the premises and the censors of the rule and collecting the facts available. The propagation phase involves, ANDing the values of the premises of the same parent, ORing the values of the censors of the same parent as the premises and finally ANDing the result of premises with the negated result of the censors. The result is then propagated to the parent, which uses it in the same manner as above till the final value of the decision is reached.

Parallelizing the evaluation of censored production rule would thus mean

- evaluating all the censors at the same level in parallel and corresponding premises in parallel, and
- traversing the rule tree in parallel to accumulate the results.

In the Parallel Censored Production Rules Model that we have simulated, the premises, censors and the propagation phases are all handled in parallel. The following subsections describe the implementation details of the model.

5.2.1 The Data Structures

Main data structures used for the implementation of the Parallel Censored Production Rules Model are the knowledge base, censor list, evaluation list and the propagation stack pointer.

Knowledge Base

For simplicity purpose we have ignored the specificities and the generalities from the rule definition. The rule can thus be called a simple Censored Production Rule. Knowledge base is implemented as an array
of records each containing the definition of the rule and the facts available as part of the knowledge. An array implementation has been chosen as the rules can be randomly accessed from it which is more time efficient and desirable for parallel models as compared to the linked list representation. The rule is defined as:

\[
\text{rule_definition} = \text{record}
\]
\[
\text{rule_name} : \text{string} ;
\]
\[
\text{rule_value} : \text{boolean} ;
\]
\[
\text{premise_list} : \text{list} ;
\]
\[
\text{censor_list} : \text{list} ;
\]
\[
\text{cf_value} : \text{array}[0..3] \text{ of real} ;
\]
\[
\text{end};
\]

Premise and censor lists contain the index of the corresponding premise and censor in the knowledge base. Thus initially the goal rule is looked for in the rule base using the rule name and all subsequent rules, corresponding to the premises and the censors, are accessed using their index values in the knowledge base.

Rule value initially has no meaning if the premise list is not empty. In case of null premise list the rule value can be TRUE or FALSE. This entry is said to correspond to a fact. Thus unless the rule entry corresponds to a fact, it will always have a valid premise list. The censor list, however, is optional. That is not all rules need have exception conditions associated with them.

Cf_array contains the certainty factors corresponding to various censor chaining depth for the rule. The index of the array corresponds to the censor chaining depth, while the value corresponds to the certainty in the decision that would be obtained if the rule has been successfully evaluated till that depth. For the implementation purpose, the rule has
been assumed to have only four levels of censor chaining depths, including the zeroth level. This is because the censors found at the upper level of the search tree have more influence on the decision than those lower down. Intuitively, deeper the assertion / censor, the more peripheral the evidence they represent. This allows for large savings in inference time at little cost in terms of precision.

All the other data structures, viz., the censor list, the evaluation list and the propagation stack pointer are pointers to the evaluation records with their individual locks.

**Evaluation Record**

The evaluation record contains the information about the rule's premises and censors that are being evaluated. The record contains the following information

```
evaluation_record = record
    rule_index : integer;
    parent_rule_index : integer;
    pc_flag : boolean;
    link_list_pointer : pointer;
end;
```

Rule_index is the index of the rule in the knowledge base. All the references of the rule are done using this index value. Parent_rule_index is the rule number of the parent to which the current premise / censor is attached. For example, if the current rule being processed is $P_{1p5}$, implying that it is the first premise of the fifth premise attached to the rule. Then the rule_index entry contains the index of the $P_{1p5}$ rule in the knowledge base and the parent_rule_index contains the index of the rule $P5$ in the knowledge base. Pc_flag indicates whether the rule is a
premise or a censor of its parent. For rule $P_{1p5}$, the $pc\_flag$ indicates that it is the premise of the rule. A link_list_pointer is maintained to keep track of the next evaluation record.

**Censor List**

All the censors encountered during the evaluation of the rule are stored in the censor list for evaluation at a later time. These censors correspond to the next censor chaining depth, and are evaluated only if the time and the censor chaining limits permit. An evaluation record is created for each censor encountered during the evaluation process. As at any time there can be more than one processors running in parallel, and thus creating evaluation records, it becomes convenient, to ensure that the same memory area is not allocated to more than one processor for creating the evaluation record. Thus for memory allocation it can be assumed that each processor has a separate heap space in the global area from which it allocates the evaluation records at run time. This removes any need for mutual exclusion for memory management.

The censor list is maintained in the global area, and as at any time there could be more than one processors trying to update the censor list, a $lock\_flag$ is set, which the processor uses to lock the censor list before attaching the evaluation record to it.

**Evaluation List**

The rules to be evaluated are picked up from the evaluation list. Thus evaluation list can be said to be the active form of the censor list. The structure of the evaluation list is same as that of the censor list. It contains a list of evaluation records, and its personal lock flag, which it uses at the time of updating the list. First the query is put on the
evaluation list. The list is then processed as follows. The first element of the list is picked up and evaluated. The censors encountered during the evaluation are put into the censors list while the premises encountered are added to the evaluation list. The evaluation process for a specific level continues until this queue is empty.

**Propagation Stack**

The propagation stack is implemented as a list of evaluation records. The rules are picked up from the evaluation list and pushed onto the propagation stack during the evaluation process. The propagation stack is traversed during the propagation phase, when all the required premises and the censors of the rule have been evaluated. The bottom of the stack contains the query, and subsequent elements contain its children. During the propagation phase, the stack elements are removed from the top. If the entry corresponds to the premise, it is ANDed with the parent entry and if it corresponds to the censor, its value is first negated and then ANDed with the parent. The process of accumulation of results continue until the stack becomes empty. The final value is thus the desired result. As there can always be more than one processors handling the propagation stack, a spinlock is provided to avoid multiple processors updating the stack at the same time.

**5.2.2 Algorithm**

The parallel evaluation procedure begins by looking for the query in the knowledge base. Once the query is found, it is placed in the evaluation list, with field rule_index containing the index of the rule in the knowledge base. As the query is at the root of the rule tree, the parent_index field and the pc_flag field of the evaluation record for the query are initialized to null. The evaluation of the rule now begins recursively in parallel. The first idle processor picks up the rule from the evaluation list and starts the evaluation process. The evaluation
record of the query is removed from the evaluation list and pushed onto the propagation stack. The censors of the rule are then extracted and placed in the censor list. These censors correspond to the next censor chaining depth of the rule. All the premises of the rule are placed in the evaluation list. This is done, so that they can be evaluated in parallel by idle processors. The premises are picked up from the evaluation list and pushed onto the propagation stack to be used later in the propagation phase. As facts out-number rules in most knowledge-bases, it is important to distinguish between them before adding the premises to the evaluation-list in the procedure ‘evaluate_a_rule’. A more straight approach would have been to put all premises in the evaluation-list. This would certainly be a simpler approach but would lead to the facts being looked at again and moved to the propagation stack in subsequent calls to ‘evaluate_a_rule’. This overhead would be very significant as the facts in any knowledge-base, especially towards the higher levels, of the rule tree are much more than the rules. By checking for the premises that are facts and adding them to the propagation stack right there minimizes this overhead. The censors are, however, always put in the censor list, irrespective of whether they are rules or facts because they would be handled at the next censor chaining depth only if the resources permit. Pushing on stack is performed by attaching the element at the head of the stack list. The process of evaluation of premises from the evaluation list continues till the list becomes empty.

The time limit and the censor chaining depth limits are checked to see if the process of evaluation for the next depth is to continue. If yes, the censors from the censor list are removed and placed in the evaluation list. This is performed by attaching the censor list to the evaluation list and making the censor list pointer null.

The evaluation process now continues for the next censor chaining depth. At the end of the evaluation process, the system starts with the propagation phase, which continues till the propagation stack becomes empty. The elements from
the stack are popped one by one, and the value of their parent is updated depending upon whether the entry corresponds to a premise or a censor, and subsequently, whether its value is true or false. As the computation time in this case is very less as compared to the time spent in processor allocation by the system, each idle processor is allocated a group of stack elements, instead of just one. Thus the back propagation of the result is handled in parallel by more than one processors.

Updating of each of the lists is an atomic process and is done by locking the corresponding list.

The algorithm for the above process is as shown below:

```plaintext
evaluate_a_rule(r,p,pcflag)
{
let (p_1,p_2,...,p_n) be premises of r;
form a lists of evaluation records
(p_i,r,TRUE) for p_i that are facts,
with (r,p,pcflag) at its head;
attach this list to propagation stack;
form a second list of evaluation records
(p_i,r,TRUE) for p_i that are rules;
attach this list to evaluation list;
let (c_1,c_2,...,c_m) be censors of r;
form a list of all (c_i,r,FALSE) evaluation records;
attach this list to censor list;
}

perform_evaluations()
{
while (evaluation_list not empty)
}
```
\{(r, p, pcflag) = extract one element from evaluation list;
if (evaluation list became empty)
{ evaluate a rule(r, p, pcflag);
unlock evaluation list;
}
else
{ unlock evaluation list;
evaluate a rule(r, p, pcflag);
}
\}

perform propagation()
{
while (propagation stack not empty)
{
let el be the current position of the top of stack (in its array);
move the stack-top down by K positions, or less if stack has less elements;
let n be the number by which the stack-top has been moved;
for i = 0 to (n-1)
{
let the value stored at location el + i be (r, p, pcflag);
update value of p from r based on whether r is premise or censor;
}
main()
{
    add (D,NULL,ϕ) to evaluation list ;
    while ( TRUE )
    {
        start all processors to do
        perform_evaluations();
        wait for all processors to get idle, i.e.
        all items in evaluation list are
        done;
        if ( evaluation to continue )
        move the contents of censor list to
        evaluation list ;
        else
        break out of while loop ;
    }
    start all processors to do
    perform_propagation();
    the value of D computed thus, is the result
}

All the underlined operations are atomic, and will therefore require the
 corresponding data structure to be locked before being executed.

5.2.3 Sample Run

To understand the algorithm, consider the following set of CPRs :

IF \( P_1 \) and \( P_2 \) and \( P_3 \)
THEN \( R_I \)
UNLESS \( C_I \) or \( C_2 \) or \( C_3 \)

IF \( P_{1p1} \) and \( P_{2p1} \)
THEN \( P_I \)
UNLESS \( C_{1p1} \) or \( C_{2p1} \) or \( C_{3p1} \)
Figure 5-2: Rule representation for the rule set of sample run
All others can be considered as facts. The rule thus has the censor chaining depth equal to 1. The same can be represented graphically as in figure 5.2. Rule R1 has a list of premises P1, P2 and P3 and a list of censors C1, C2 and C3 attached to it. P1 is again a rule with two premises and three censors. Figure 5.3 shows the data structures at various steps, during the evaluation process. Let the query be for rule R1. The query is first searched for in the knowledge base, and then placed in the evaluation list (Fig 5.3(a)). The censor list and the propagation stack are empty at this time. In step two (Fig 5.3(b)), the rule from the evaluation list is placed in the propagation stack, the premises of the rule are added to the evaluation list and the censors to the censor list. The premises which are facts, are directly attached to the propagation stack. This corresponds to the evaluation of the rule for censor chaining depth zero. At the end of evaluation for that level, the evaluation list becomes empty, the censor list contains the censors corresponding to the next censor chaining depth, and the propagation stack contains the rules just evaluated (Fig 5.3©). For the evaluation of the rule for next censor chaining depth, the censor list is attached to the evaluation list, censor list becomes null (Fig 5.3(d)) and the process of evaluation continues from step 2.

5.2.4 Salient Features of the Algorithm

Following are the salient features of the algorithm that make it very efficient

1. Parallelism Of Premises, Censors and Propagation Phase

The algorithm ensures high level of parallelism, as in this the premises as well as the censors are handled in parallel. In addition to this the back propagation of the results is being done by more than one processors in parallel. Thus except for a few atomic operations, which involve updation of the lists (evaluation, censor and the propagation stack) most of the other operations in the algorithm can be evaluated by more
Figure 5-3: Data structures during various steps of execution of censored production rules (a) At the start of evaluation of query (b) Evaluation of rule at censor chaining depth 0
Step III

**Evaluation List**

![Evaluation List Diagram]

**Censor List**

![Censor List Diagram]

**Propogation Stack**

![Propagation Stack Diagram]

(c)

*Figure 5-3 (c): At the end of evaluation of the rule at current censor chaining depth*
Figure 5-3 (d): At the start of evaluation of the rule at next censor chaining depth
than one processors at a time.

2. Load Balancing

The algorithm handles load balancing problem by forming a work pool of all the rules that need to be handled. Evaluation list and the propagation stack act as work pools during the evaluation phase and the propagation phases respectively. As and when any processor becomes idle, it looks for more work in the work pool. If there are any premises to be evaluated, it picks up that from the evaluation list and starts the evaluation. Similarly during the propagation phase, the idle processors look for work in the propagation stack and pick up the rules from the stack, whose results need to be back propagated.

3. Minimization Of Contention

The contention problem in the algorithm is handled in the following manner.

- Attaching the premises which are facts directly to the propagation stack. An easier and a general approach would have been to attach all premises, irrespective of whether they are facts or rules to the evaluation list, and later during the evaluation process, if they are rules, extract their premises and censors and attach them to the corresponding lists, and in case of facts attach them to the propagation stack. Thus for the facts, one notes, that evaluation simply means, attaching the corresponding evaluation record first to the evaluation list and later to the propagation stack without doing anything extra with it. As for most practical knowledge bases, most of the premises turn out to be facts at higher rule tree heights, attaching the facts first to the evaluation list and then to the
propagation stack can lead to severe bottleneck. The problem is handled by checking whether the premise is a rule or a fact before starting with its evaluation and then subsequently attaching them to appropriate lists. This also minimizes the processor contention.

• **Lists instead of arrays.** The evaluation list, the censor list and the propagation stack are all implemented as lists. Attaching an element to any of this list would thus mean updating two pointers only, rather than copying the complete record field by field. This reduces the total number of instructions to be executed thereby improving the algorithm performance.

• **Grouping of tasks in the propagation stack.** Propagation stack consists of rules, whose values have to be accumulated, depending upon whether they correspond to a premise or a censor, and the result propagated back to its parent. Thus the complete operation involves ANDing the value of the rule directly or after negating it to its parent, depending upon whether it corresponds to a premise or a censor respectively. The total time for computing the value is much less than the time spent in allocating the task to the idle processor. To reduce the contention, the tasks in the propagation stack are grouped together before being allocated to the processor. Thus, the processor gets a group of tasks (say 10) instead of one for processing.

4. **Partial ordering amongst the rules on the stack.**

The evaluation process starts from the root going down the rule tree step by step. The premises are handled depth first, and the corresponding censors are evaluated only after their premises have been moved to the propagation stack. Thus, even though the premises are
being evaluated in parallel, on the propagation stack, partial ordering among the rules, such that all the premises of the rule appear higher up in the stack than its parent is ensured. This guarantees, that during the back propagation phase, the value of the rule will be fully updated before it is used.

5.3 Results and Discussions

For the measurement purpose, three sample knowledge bases were considered. The first knowledge base (CASE I) consisted of about 10000 rules, with rules having typically five premises and five censors each. In the second knowledge base (CASE II), the rules consisted of five premises and three censors each, while in the third (CASE III) the number of premises associated with each were eight and the number of censors three. All the three knowledge bases had censor chaining depth equal to 4. The number of rules and facts for each censor chaining depth were found to be as shown in Table 5-1:

<table>
<thead>
<tr>
<th>Level</th>
<th>CASE I</th>
<th>CASE II</th>
<th>CASE III</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rules</td>
<td>Facts</td>
<td>Rules</td>
</tr>
<tr>
<td>0</td>
<td>156</td>
<td>625</td>
<td>156</td>
</tr>
<tr>
<td>1</td>
<td>586</td>
<td>3125</td>
<td>414</td>
</tr>
<tr>
<td>2</td>
<td>986</td>
<td>6875</td>
<td>558</td>
</tr>
<tr>
<td>3</td>
<td>1111</td>
<td>9375</td>
<td>585</td>
</tr>
</tbody>
</table>

Table 5-1: Number of rules and facts for different cases used for the experimentation of Parallel Censored Production Rules model

Speedups were measured for various censor chaining depths for different number of processors. Tables 5-2, 5-3 and 5-4 show the units of time taken by
Table 5-2: Time taken by P processors to evaluate a rule upto various censor chaining depths for CASE I

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Depth 0</th>
<th>Depth 1</th>
<th>Depth 2</th>
<th>Depth 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>152578</td>
<td>710156</td>
<td>1483806</td>
<td>1963050</td>
</tr>
<tr>
<td>2</td>
<td>76672</td>
<td>355078</td>
<td>741903</td>
<td>981525</td>
</tr>
<tr>
<td>3</td>
<td>51201</td>
<td>237510</td>
<td>494604</td>
<td>654350</td>
</tr>
<tr>
<td>4</td>
<td>39123</td>
<td>178881</td>
<td>371881</td>
<td>491993</td>
</tr>
<tr>
<td>6</td>
<td>27492</td>
<td>119555</td>
<td>249379</td>
<td>328819</td>
</tr>
<tr>
<td>8</td>
<td>23546</td>
<td>100446</td>
<td>210171</td>
<td>278052</td>
</tr>
<tr>
<td>10</td>
<td>22405</td>
<td>96884</td>
<td>202429</td>
<td>268543</td>
</tr>
<tr>
<td>15</td>
<td>20398</td>
<td>91397</td>
<td>193456</td>
<td>257618</td>
</tr>
<tr>
<td>18</td>
<td>19738</td>
<td>88992</td>
<td>188779</td>
<td>253952</td>
</tr>
<tr>
<td>20</td>
<td>19289</td>
<td>87891</td>
<td>187113</td>
<td>251029</td>
</tr>
</tbody>
</table>
Table 5-3: Time taken by P processors to evaluate a rule up to various censor chaining depths for CASE II

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Depth 0</th>
<th>Depth 1</th>
<th>Depth 2</th>
<th>Depth 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>145626</td>
<td>468820</td>
<td>2851782</td>
<td>3014224</td>
</tr>
<tr>
<td>2</td>
<td>73179</td>
<td>234410</td>
<td>370501</td>
<td>421585</td>
</tr>
<tr>
<td>3</td>
<td>48868</td>
<td>156796</td>
<td>247827</td>
<td>281997</td>
</tr>
<tr>
<td>4</td>
<td>37340</td>
<td>118091</td>
<td>186181</td>
<td>211852</td>
</tr>
<tr>
<td>6</td>
<td>26145</td>
<td>79867</td>
<td>125381</td>
<td>141709</td>
</tr>
<tr>
<td>8</td>
<td>22439</td>
<td>67749</td>
<td>105857</td>
<td>119769</td>
</tr>
<tr>
<td>10</td>
<td>21574</td>
<td>64844</td>
<td>102917</td>
<td>116783</td>
</tr>
<tr>
<td>15</td>
<td>19949</td>
<td>61850</td>
<td>98146</td>
<td>111975</td>
</tr>
<tr>
<td>18</td>
<td>19365</td>
<td>60649</td>
<td>96109</td>
<td>109931</td>
</tr>
<tr>
<td>20</td>
<td>18937</td>
<td>59646</td>
<td>95245</td>
<td>108796</td>
</tr>
</tbody>
</table>
Table 5-4: Time taken by P processors to evaluate a rule upto various censor chaining depths for CASE III

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>Depth 0</th>
<th>Depth 1</th>
<th>Depth 2</th>
<th>Depth 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>896528</td>
<td>2170884</td>
<td>2851782</td>
<td>3014224</td>
</tr>
<tr>
<td>2</td>
<td>448264</td>
<td>1085442</td>
<td>1425891</td>
<td>1507112</td>
</tr>
<tr>
<td>3</td>
<td>299843</td>
<td>723628</td>
<td>950594</td>
<td>1004741</td>
</tr>
<tr>
<td>4</td>
<td>225258</td>
<td>544081</td>
<td>714732</td>
<td>755445</td>
</tr>
<tr>
<td>6</td>
<td>150677</td>
<td>363632</td>
<td>478487</td>
<td>504051</td>
</tr>
<tr>
<td>8</td>
<td>126807</td>
<td>309684</td>
<td>402794</td>
<td>425137</td>
</tr>
<tr>
<td>10</td>
<td>122812</td>
<td>296164</td>
<td>391728</td>
<td>413474</td>
</tr>
<tr>
<td>15</td>
<td>116584</td>
<td>285643</td>
<td>375235</td>
<td>397655</td>
</tr>
<tr>
<td>18</td>
<td>113485</td>
<td>277962</td>
<td>368924</td>
<td>390950</td>
</tr>
<tr>
<td>20</td>
<td>112206</td>
<td>274795</td>
<td>365613</td>
<td>387432</td>
</tr>
</tbody>
</table>
P processors to evaluate a rule upto various censor chaining depths for three knowledge bases. The same is represented graphically in Figures 5.4, 5.5, and 5.6. The speedups in each case were found to be consistent. These are shown in figure 5.7. The graphs indicate that in each case the time taken to reach up to a specific censor chaining depth decreases almost linearly up to six processors, after which it becomes very gradual for up to eight processors. Beyond eight processor the time taken is almost constant. This is evident from fig 5.7 also, where the speedups increase almost linearly up to 6 processor, after which they become very gradual. The speedups are almost constant from number of processors beyond 8.

The main reason for slow speedups after 6 processors is due to processor contention. There are three main data structures, viz., the evaluation list, the censor list and the propagation stack in the shared area, which are being accessed by all the processes. To prevent the corruption of data due to concurrent access, these need to be locked before being accessed. Hence the processors have to some time wait for the shared data structure to be unlocked before accessing it. This leads to processor contention, thereby reducing the speedups and the processor efficiency (fig 5.8).

The contention problem has been dealt with to a large extent by creating a workpool of premises and censors in the evaluation phase, and later in the propagation phase by grouping the tasks, such that the locking can be minimized. This has helped in balancing the work load amongst the processors such that the processor efficiency is almost hundred percent up to six processors.
Figure 5-4: Time taken by P processors to evaluate the rule upto various censor chaining depths for CASE I.
Figure 5-5: Time taken by P processors to evaluate the rule up to various censor chaining depths for CASE II.
Figure 5-6: Time taken by P processors to evaluate the rule upto various censor chaining depths for CASE III.
Figure 5-7: Speedups obtained from Parallel Censored Production Rules model
Figure 5-8: Processor utilization by Parallel Censored Production Rules model