Efficient Anticipatory Reasoning for Anticipatory Systems with Requirements of High Reliability and High Security

Yuichi Goto, Shinsuke Nara, and Jingde Cheng
Department of Information and Computer Sciences, Saitama University
Saitama, 338-8570, Japan
{gotoh, nara, cheng}@aise.ics.saitama-u.ac.jp

Abstract A practical anticipatory system with requirements of high reliability and high security must be able to perform any anticipatory reasoning to get enough effective conclusions anticipatorily within an acceptable time in order to satisfy the requirements from applications. This is a contradictory requirement since the execution time of anticipatory reasoning gets longer in proportion to the amount increasement of deduced conclusions. We are developing a forward deduction system for general-purpose entailment calculus, named EnCal. Although EnCal is a forward deduction engine for general-purpose entailment calculus, we expect that it can serve as the forward deduction engine in an anticipatory system to perform anticipatory reasoning based on temporal relevant logics. The key issue to achieve this goal is the efficiency of EnCal. This paper presents results and their implications of our experiences on improving the efficiency of EnCal by parallel processing techniques.
Keywords : Highly reliable systems, Highly secure systems, Anticipation, Anticipatory reasoning, Forward deduction for entailment calculus, Parallel processing.

1 Introduction

Reasoning is the process of drawing new conclusions from given premises, which are already known facts or previously assumed hypotheses. An anticipatory reasoning is a reasoning to draw new, previously unknown and/or unrecognized conclusions about some future event or events whose occurrence and truth are uncertain at the point of time when the reasoning is being performed. Obviously, any anticipatory reasoning must be forward rather than backward. From the philosophical viewpoint, the notion of anticipation itself is intrinsically time dependent. The earlier an anticipatory reasoning draws conclusions, or the farther the future event is predicted by an anticipatory reasoning, the higher is its degree of anticipation. To be a computing system useful in various applications in the real world, an anticipatory system must have the ability of anticipatory reasoning with some certain degree of anticipation to predict the occurrence and truth of some future event or events.

International Journal of Computing Anticipatory Systems, Volume 14, 2004
Edited by D. M. Dubois, CHAOS, Lìege, Belgium, ISSN 1373-5411 ISBN 2-930396-00-8

On the other hand, from the viewpoints of software reliability engineering and information security engineering, a practical anticipatory system must be able to perform any anticipatory reasoning to get enough effective conclusions anticipatorily within an acceptable time in order to satisfy the requirements of high reliability and high security from applications. Since the most intrinsic characteristic of an anticipatory system is its ability of taking anticipation, an anticipatory system that cannot satisfy the requirements of anticipation and timeliness is useless at all in practices in the real world. Therefore, for an anticipatory system with requirements of high reliability and high security, its functioning is both anticipation-critical and time-critical.

Thus, we face a dilemma. On the one hand, as a forward reasoning, an anticipatory reasoning has to deal with a lot of intermediates, which are usually involved in any forward reasoning, in order to get effective conclusions anticipatorily. On the other hand, an anticipatory reasoning has to be performed as efficiently as possible in order to keep a high degree of anticipation.

We are developing a forward deduction system for general-purpose entailment calculus, named EnCal. Although EnCal is a forward deduction engine for general-purpose entailment calculus, we expect that it can serve as the forward deduction engine in an anticipatory system to perform anticipatory reasoning based on temporal relevant logics [7]. The key issue to achieve this goal is the efficiency of EnCal. This paper presents results and their implications of our experiences on improving the efficiency of EnCal by parallel processing techniques.

The rest of this paper is organized as follows: Section 2 explains the position of our research. Section 3 gives an analysis for execution time of a forward deduction engine for anticipatory reasoning. Section 4 gives some explanations about EnCal and presents a model of its parallelization version in order to show a case study of improving the performance of a forward deduction engine for anticipatory reasoning by parallel processing. Section 5 shows our implementation of the parallelization version of EnCal and our experiments on a shared-memory parallel computer and clusters of PCs. Section 6 discusses our experimental results. Some concluding remarks are given in Section 7.

2 Forward deduction engine for anticipatory reasoning

The concept of an anticipatory system first proposed by Rosen in 1980s [16]. Rosen considered that "an anticipatory system is one in which present change of state depends upon future circumstance, rather than merely on the present or past" and defined an anticipatory system as "a system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's prediction to a latter instant." [16] Until now, philosophical discussions on anticipatory systems and their characteristics are still being continued by scientists from various disciplines [8, 9, 10, 12, 13].

On the other hand, from the viewpoints of software reliability engineering and information security engineering, what we need is really useful systems with anticipatorily predictive capability to take anticipation for forestalling disasters and attacks rather than the philosophical definition and intension of an anticipatory system. In order to develop anticipatory systems useful in the real world, we has proposed a new type of reactive systems, named "Anticipatory Reasoning-Reacting Systems," as a certain class of anticipatory systems [5].

An anticipatory reasoning-reacting system (ARRS for short) is a computing system containing a controller C with capabilities to measure and monitor the behavior of the whole system, a traditional reactive system RS, a predictive model PM of RS and its external computing environment, and an anticipatory reasoning engine ARE such that according to predictions by ARE based on PM, C can order and control RS to carry out some operations with a high priority [5]. In this paper, we discuss about anticipatory reasoning engine ARE.

An ARE must be a forward deduction engine. Reasoning can be classified into forward reasoning and backward reasoning. Forward reasoning is to infer new conclusions from known facts or assumed hypotheses. Backward reasoning is to find out the path which is from known facts or hypotheses to given goal or sub-goal. Anticipatory reasoning is forward rather than backward because when we perform an anticipatory reasoning we cannot know some future event or events, whose occurrence and truth are uncertain at the time point of the reasoning is being performed, as a goal or sub-goal. Reasoning can be classified into three forms, deduction, induction and abduction. Deduction is the process of deducing or drawing a conclusion from some general principles already known or assumed. Induction is the process of inferring some general laws or principles from the observation of particular instances. Abduction is the process whereby a surprising fact is made explicable by the application to it of a suitable proposition. For an ARRS, the conclusions deduced by the ARE must be definitely correct if the premises are correct. This can be guaranteed by only deduction. Therefore, an ARE must be a forward deduction engine.

3 Computational complexity of forward deduction for anticipatory reasoning

A forward deduction engine for anticipatory reasoning, as well as other forward deduction engines, has a difficult problem that its execution time gets longer in proportion to the increasement of deduced conclusions. In general, a forward deduction engine performs the following four processes repeatedly for each inference rule, in order to get new conclusions, until some termination conditions are satisfied.

1. Matching process: it seeks and picks up some premises, which are to be applied to an inference rule, from a set of premises. Then it matches the premises to an inference rule.

2. Deduction process: it applies an inference rule to the premises which were matched at the matching process.
3. Duplication checking process: it compares a conclusion which was deduced at the deduction process with all previously deduced conclusions and premises in order to check whether it is a duplicate or not.
4. Adding process: it adds the conclusion which was judged to be new at the duplication checking process to the set of premises.

We present some equations about the execution time and the amount of data of a forward deduction engine if all inference rules apply to all given premises. Let n be the number of previously given premises, I be the number of inference rules, r be the number of premises required by an inference rule, in this assumption all inference rules require r premises, and τ_m be the execution time of judging whether a inference rule can apply to some premises or not and matching the inference rule to the premises at the matching process. The execution time at the matching process is

$$n^r \cdot I \cdot \tau_m. \tag{1}$$

Let τ_d be the execution time of deducing a conclusion. The execution time at the deduction process is at most

$$n^r \cdot I \cdot \tau_d. \tag{2}$$

Let τ_c be the execution time of comparing a deduced conclusion at the deduction process with a previously deduced conclusion or a given premise at the duplication checking process. The execution time at the duplication checking process is at most

$$\begin{aligned} & n \cdot I \cdot \tau_c + (1 \cdot \tau_c + 2 \cdot \tau_c + \ldots + (I \cdot n^r - 1) \cdot \tau_c) \\ = \; & (n \cdot I + \sum_{k=1}^{I \cdot n^r - 1} k) \cdot \tau_c \\ = \; & \frac{1}{2}\{I^2(n^r - 1)^2 + I(n^r + 2n - 1)\} \cdot \tau_c. \end{aligned} \tag{3}$$

In this assumption, the total execution time of above three process is

$$\begin{aligned} & n^r \cdot I \cdot (\tau_m + \tau_d) + \frac{1}{2}\{I^2(n^r - 1)^2 + I(n^r + 2n - 1)\} \cdot \tau_c \\ \approx \; & O(I^2 \cdot n^{2r}). \end{aligned} \tag{4}$$

On the other hand, the number of deduced conclusions in forward deduction is huge. In this assumption, the number of deduced conclusions is at most

$$I \cdot n^r. \tag{5}$$

In the adding process, deduced new conclusions are added into a set of premises. Then processing of forward deduction is again repeated using the conclusions as premises. Thus, even the number of premises is few, the number of deduced conclusions become large easily. We therefore can regard the execution time of a forward deduction engine as $O(N^{2r})$, where N denotes the number of data which includes both finally deduced conclusions and given premises, because the number of data is rather large than the number of given inference rules.

A useful forward deduction engine must get enough effective conclusions in an acceptable time. This is especially true to a forward deduction engine for anticipatory reasoning. Thus this problem must be solved in order to implement an efficient forward deduction engine as an anticipatory reasoning engine. The efficiency of a forward deduction engine can be improved by two aspects. One is shortening the execution time of each process in a forward deduction engine. Another is reducing the processing load. The first aspect focuses on the execution time of each process, i.e., shortening τ_m, τ_d and τ_c. It is expected that it can shorten the execution time of a forward deduction engine at a constant rate without the increasement in the number of deduced conclusions and given premises. The second aspect is focuses on the processing load: the order of the execution time become less than $O(N^{2r})$ where N is the number of deduced conclusions and given premises and r is the number of premises required by an inference rule. This aspect can be classified into two approaches. One is to narrow down the range of data being processed. As at a certain time, all data is not necessarily performed on a certain process of a forward deduction engine. It is also expected that this approach can shorten the execution time at a constant rate without the increasement in the number of deduced conclusions and given premises. Other is reducing the processing load on one processor by parallel processing. It is expected that this approach can shorten the execution time in proportion to the number of using processors.

In this paper, we focus on reducing the processing load on one processor by parallel processing. This approach is flexible to the increasement in the number of deduced conclusions and given premises since it can increase the number of processors. In following sections, we present a case study of improving the performance of an automated forward deduction system for general-purpose entailment calculus, named EnCal, as an anticipatory reasoning engine with parallel processing.

4 The case study of EnCal

We are developing an automated forward deduction system for general-purpose entailment calculus, named EnCal [4]. Although EnCal is a forward deduction engine for general-purpose entailment calculus, we expect that it can serve as the forward reasoning engine to perform anticipatory reasoning based on temporal relevant logics in an anticipatory system [7].

4.1 Forward deduction for entailment calculus

An *entailment calculi* is a formalization of a logical system L such that the notion of conditional (entailment) is represented in L by a primitive connective and all logical theorems of L are represented in the form of entailment.

In logic, a sentence in the form of "if ... then ..." is usually called a conditional proposition or simply conditional. A conditional must concern two parts which are connected by the connective "if ... then ..." and called the antecedent and the consequent of that conditional. The truth of a conditional depends not only on the truth of its antecedent and consequent but also, and more essentially, on a necessarily relevant and/or conditional relation between its antecedent and consequent.

When we study and use logic, the notion of conditional may appear in both the object logic (i.e., the logic we are studying) and the meta-logic (i.e., the logic we are using to study the object logic). From the viewpoint of the object logic, there are two classes of conditionals. One class is empirical conditionals and the other class is logical conditionals. In the sense of logic, an empirical conditional is that its truth-value is depend on the contents of its antecedent and consequent. Therefore the truth-value of an empirical conditional cannot be determined only by its abstract form. A logical conditional is that its truth-value is universally true or false and therefore can be determined by its abstract form. A logical conditional that is considered to be universally true, in the sense of that logic, is also called entailment of that logic [1, 2, 6].

A formal logic system L consists of a formal language, called the object language and denoted by $F(L)$, which is the set of all well-formed formulas of L, and a logical consequence relation, denoted by meta-linguistic symbol $\vdash_L$, such that $P \subseteq F(L)$ and $c \in F(L)$, $P \vdash_L c$ means that within the frame work of L, c is valid conclusion of premises P, i.e., c validly follows from P. For a formal logic system $(F(L), \vdash_L)$, a logical theorem t is a formula of L such that $\phi \vdash_L t$ where ϕ is empty set. We use $Th(L)$ to denote the set of all logical theorems of L.

Let $(F(L), \vdash_L)$ be a formal logic system and $P \subseteq F(L)$ be a non-empty set of sentences (i.e., closed well-formed formulas). A formal theory with premises P based on L, called a L-theory with premises P and denoted by $T_L(P)$, is defined as $T_L(P) =_{df} Th(L) \cup Th^e_L(P)$, and $Th^e_L(P) =_{df} \{\text{et} | P \vdash_L \text{et and et} \notin Th(L)\}$ where $Th(L)$ and $Th^e_L(P)$ are called the logical part and the empirical part of the formal theory, respectively, and any element of $Th^e_L(P)$ is called an empirical theorem of the formal theory.

For a formal logic system where the notion of conditional is represented by primitive connective entailment "$\Rightarrow$", a formula is called a zero degree formula if and only if there is no occurrence of "$\Rightarrow$" in it; a formula of the form "$A \Rightarrow B$" is called a first degree conditional if and only if both A and B are zero degree formula; a formula A is called a first degree formula if and only if it satisfies one of the following conditions:

1. A is a first degree conditional,

2. A is in the form $+B$ ($+$ is a one-place connective such as negation and so on) where B is a first degree formula,
3. A is in the form $B * C$, ($*$ is a non-implicational two-place connective such as conjunction or disjunction and so on), where both of B and C is a first degree formulas, or one of B and C are a first degree formula and the another is a zero degree formula.

Let k be a natural number. A formula of the form "$A \Rightarrow B$" is called a k^{th} degree conditional if and only if both A and B are $(k-1)^{th}$ degree formulas, or either formula A or B is a $(k-1)^{th}$ degree formula and the another is a $j^{th}(j < k-1)$ degree formula; a formula is called k^{th} degree formula if and only if it satisfies one of the following conditions:

1. A is a k^{th} degree conditional,
2. A is in the form $+B$ ($+$ is a one-place connective such as negation and so on) where B is a k^{th} degree formula,
3. A is in the form $B * C$, ($*$ is a non-implicational two-place connective such as conjunction or disjunction and so on), where both of B and C is a k^{th} degree formulas, or one of B and C are a k^{th} degree formula and the another is a $j^{th}(j < k)$ degree formula.

Let $(F(L), \vdash_L)$ be a formal logic system and k be a natural number. The k^{th} degree fragment of L, denoted by $Th^k(L)$, is a set of logical theorems of L that is inductively defined as follows (in the terms of Hilbert-style formal systems):

1. if A is a $j^{th}(j \leq k)$ degree formula and an axiom of L, then $A \in Th^k(L)$,
2. if A is a $j^{th}(j \leq k)$ degree formula that is the result of applying an inference rule of L to some members of $Th^k(L)$, then $A \in Th^k(L)$,
3. nothing else is a member of $Th^k(L)$, i.e., only those obtained from repeated applications of 1. and 2. are members of $Th^k(L)$.

Let $(F(L), \vdash_L)$ be a formal logic system, $P \subset F(L)$, and k and j be two natural numbers. A formula A is said to be j^{th}-degree-deducible from P based on $Th^k(L)$ if and only if there is an finite sequence of formulas $f_1, \ldots, f_n$ such that $f_n = A$ and for all $i(i \leq n)$ (1) $f_i \in Th^k(L)$, or (2) $f_i \in P$, or (3) f_i whose degree is not higher than j is the result of applying an inference rule to some members $f_{j_1}, \ldots, f_{j_m}(j_1, \ldots, j_m < i)$ of the sequence. If $P \neq \phi$, then the set of all formulas which are j^{th}-degree-deducible from P based on $Th^k(L)$ is called the j^{th} degree fragment of the formal theory with premises P based on $Th^k(L)$, denoted by $T^j_{Th^k(L)}(P)$ [6].

Automated forward deduction is a process of deducing new and unknown conclusions automatically by applying inference rules to premises and previously deduced conclusions repeatedly until some previously specified condition is satisfied.

4.2 EnCal

EnCal supports an automated forward deduction for entailment calculi based on strong relevant logics as well as other logics [4]. It provides its users with the following major facilities. For a formal logic system L which may be a propositional logic, a first-order predicate logic, or a second-order predicate logic, a non-empty set P of formulas as premises, inference rules of logic system L and natural number k and j (usually, $k, j \leq 5$) as limit of degree which is the degree of nested entailment (denoted by "$\Rightarrow$" in this paper), all specified by the user, EnCal can

1. reason out all logical theorem schemata of the $Th^k(L)$,
2. verify whether or not a formula is a logical theorem schema of the $Th^k(L)$, if yes, then give the proof,
3. reason out all empirical theorems of the j^{th} degree fragment of L-theory with premises P based on $Th^k(L)$,
4. verify whether or not a formula is an empirical theorem of the j^{th} degree fragment of L-theory with premises P based on $Th^k(L)$, if yes, then give the proof [4].

In this paper, we focus on the function that reasons out all logical theorem schemata, LTSs for short, of the k^{th} degree fragment of a propositional logic since it is most basic function of EnCal.

An automated forward deduction by EnCal consists of 3 parts as follows.

1. Initialization: it takes in premises, limits of degree k and j and inference rules.
2. Forward deduction: about each inference rule, it repeats following processes until it deduces no new LTSs.
   (a) Matching process: it seeks and picks up some LTSs from a set of previously deduced LTSs or premises to apply an inference rule. Then it matches the LTS to the inference rule.
   (b) Deduction process: it applies the inference rule to the LTSs which were matched at the matching process.
   (c) Duplication checking process: it compares a conclusion which was deduced at the deduction process with all previously deduced LTSs in order to check whether it is duplicate or not.
   (d) Adding process: it adds the conclusion which was judged to be new at the duplication checking process as new LTS to the set of premises if the its degree of nest of entailment is l^{th} degree ($1 \leq l \leq k$).
3. Outputting: it outputs all deduced LTSs to a file.

At present, the inference rule of EnCal is Modus Ponens only. Modus Ponens is that B is deduced from $A \Rightarrow B$ and A.

4.3 Computational complexity analysis

The each process in the forward deduction part is depends on the result of previous process prior to it.

The forward deduction algorithm in EnCal is as follows. Let n be the number of previously deduced LTSs and given premises, and $\{P\} = \{P_0, P_1, \ldots, P_{n-1}\}$ be the set of premises and previously deduced LTSs.

Algorithm 1 Forward deduction

1. $n \leftarrow$ the number of premises.
2. $p \leftarrow 0$
3. $k \leftarrow$ the limit of degree.
4. **do**
5. $\quad n^{'} \leftarrow n$
6. $\quad$ **for** ($i \leftarrow 0$, $i < n$, $i \leftarrow i + 1$)
7. $\quad\quad$ **for** ($j \leftarrow p$, $j < n$, $j \leftarrow j + 1$)
8. $\quad\quad\quad$ Matching(P_i,P_j):
   If it can apply Modus Ponens to between P_i and P_j, return SUCCESS.
   If no, return FAILURE.
9. $\quad\quad\quad$ **if** Matching(P_i,P_j) returns SUCCESS
10. $\quad\quad\quad\quad$ **then** Deduction(P_i,P_j):
    it applies Modus Ponens to P_i and P_j.
11. $\quad\quad\quad\quad$ Duplication_check(C):
    If a conclusion C which was deduced at Deduction(P_i,P_j) is duplicate, return DUPLICATE. If no, return NEW.
12. $\quad\quad\quad\quad$ **if** Duplication_check(C) returns NEW
13. $\quad\quad\quad\quad\quad$ **then** Adding(C):
    it adds an a conclusion C into $\{P\}$ if the degree of C is smaller than k. After that $n' \leftarrow n^{'} + 1$.
14. $\quad$ **for** ($i \leftarrow p$, $i < n$, $i \leftarrow i + 1$)
15. $\quad\quad$ **for** ($j \leftarrow 0$, $j < p$, $j \leftarrow j + 1$)
16. $\quad\quad\quad$ Matching(P_i,P_j)
17. $\quad\quad\quad$ **if** Matching(P_i,P_j) returns SUCCESS
18. $\quad\quad\quad\quad$ **then** Deduction(P_i,P_j)
19. $\quad\quad\quad\quad$ Duplication_check(C)
20. $\quad\quad\quad\quad$ **if** Duplication_check(C) returns NEW
21. $\quad\quad\quad\quad\quad$ **then** Adding(C)
22. $\quad p \leftarrow n$
23. $\quad n \leftarrow n^{'}$
24. **while** (new LTSs are deduced).

The algorithm of Duplication_check(C) is as follows. Let Comp(A,B) be a function which compares A with B to judge whether B is duplicate of A. If B is duplicate, then Comp(A,B) returns DUPLICATE.

Algorithm 2 Duplication_check(C)

1. **for** ($i \leftarrow 0$, $i < n$, $i \leftarrow i + 1$)
2. Comp(P_i,C):
   C is a conclusion which was deduced at Deduction process.
3. **if** Comp(P_i,C) returns DUPLICATE
4. **then return** DUPLICATE
5. **return** NEW.

The major portion of the execution time of EnCal is spent at the duplication checking process. Let N be the number of given premises and new LTSs which are deduced finally. The number of premises required by Modus Ponens is two. The number of times of processing Matching(A,B) is N^2. The number of times of processing Duplication_check(A) is at most N^2 and at least N, since the number of deduced conclusions and intermediates is at most N^2 and at least N. The number of times of processing Comp(A,B) is at most

$$\sum_{k=1}^{N^2-1} k = \frac{\{N^2(N^2-1)\}}{2}, \tag{6}$$

and at least

$$\sum_{k=1}^{N-1} k = \frac{\{N(N-1)\}}{2}. \tag{7}$$

Thus the calculated amount of EnCal approaches at most $O(N^4)$ and at least $O(N^2)$. EnCal also has a problem that its execution time gets longer in proportion to the amount increasement of deduced conclusions.

4.4 Parallel version of EnCal

We summarize the processing features of EnCal.

1. The each process in the forward deduction part depend on the results of previous process prior to it.
2. Previously deduced LTSs are accessed frequently at the matching process and the duplication checking process.
3. The number of times of processing Matching(A,B) is at most N^2, where N is the number of finally deduced LTSs and given premises.
4. The number of times of processing Comp(A,B) is at most $\{N^2(N^2-1)\}/2$ and at least $\{N(N-1)\}/2$.

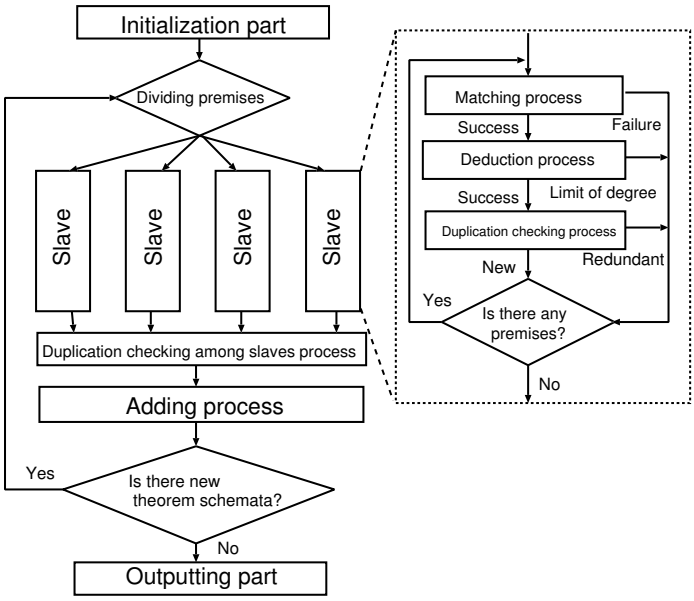

Fig. 1: The model of parallelization version of EnCal

We design the parallelization version of EnCal based on master-slave model. Master-slave model based on agenda parallelism paradigm [3] is a suitable model for parallelization version of EnCal because of above features 1, 3 and 4. Figure 1 shows the model of parallelization version of EnCal. The model consists of following parts where the initialization part, the matching process, the deduction process, the duplication checking process, the adding process and the outputting part are same one of the sequential version of EnCal.

1. Initialization.
2. Forward deduction: about each inference rule, it repeats following processes until it deduces no new LTSs.
   (a) Master: it changes a slave part after dividing premises to other slaves.
   (b) Slave: the following processes are repeated independently of other slaves.
      i. Matching process.
      ii. Deduction process.
      iii. Duplication checking process.
   (c) Duplication checking among slaves process: it detects and reduces the duplicate which is not detected at the duplication checking process in Slave part.
   (d) Adding process.
3. Outputting.

It is necessary for efficient duplication check to be able to access all previously deduced LTSs. However, the set of deduced conclusions at a slave is not referred by other slaves. The parallelization of EnCal needs the duplication checking among slaves part.

On this parallelization of EnCal, let p be the number of processors, and N be the number of given premises and new LTSs which are deduced finally. If deduced conclusions or intermediates are evenly deduced on each slave, the number of times of processing Comp(A,B) at the duplication check process on one slave is at most

$$\frac{\sum_{k=1}^{N^2-1} k}{p} = \frac{\{N^2(N^2-1)\}}{2p}, \tag{8}$$

and at least

$$\frac{\sum_{k=1}^{N-1} k}{p} = \frac{\{N(N-1)\}}{2p}. \tag{9}$$

The number of times of processing Comp(A,B) at the duplication checking among slaves process is at most

$$p \cdot \sum_{k=1}^{N^2/p-1} k = \frac{\{N^2(N^2-p)\}}{2p}, \tag{10}$$

and at least

$$p \cdot \sum_{k=1}^{N/p-1} k = \frac{\{N(N-p)\}}{2p}. \tag{11}$$

Thus the theoretical speed up ratio is approximated as follows,

$$\begin{aligned} \text{Speed up ratio} &\approx \frac{\frac{\{N^2(N^2-1)\}}{2p} + \frac{\{N^2(N^2-p)\}}{2p}}{\frac{\{N^2(N^2-1)\}}{2}} \quad \text{(at most)} \\ &\approx \frac{\frac{\{N(N-1)\}}{2p} + \frac{\{N(N-p)\}}{2p}}{\frac{\{N(N-1)\}}{2}} \quad \text{(at least)} \\ &\approx \frac{2}{p} \end{aligned} \tag{12}$$

In the parallelization version of EnCal based on master-slave model, its theoretical value shows that the processing load on one processor decreases in proportion to the increasement in used processors.

Table 1: The execution time on Sun Enterprise 6000 (sec)

Logic systems	1 processor	2 processors	4 processors	8 processors	16 processors
Te(4)	3499	1641	922	562	381
Ee(4)	12347	6100	3370	2021	1290
Re(4)	85962	41146	21825	12668	8236

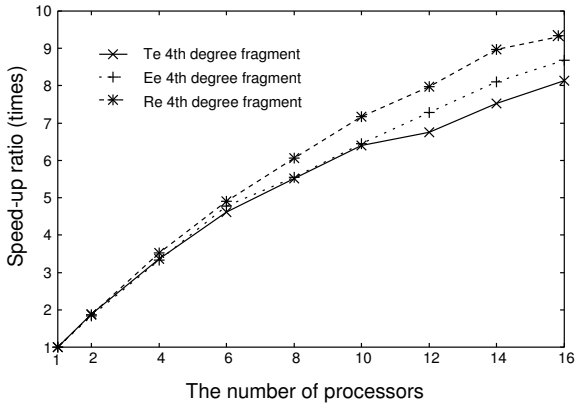

Fig. 2: Speed-up ratio on Sun Enterprise 6000

5 Implementation and Results

We have implemented the parallelization version of EnCal based on master-slave model on a shared-memory parallel computer and a cluster of PCs, and got the execution time of deducing 4th degree fragment from axioms of some logic systems, in order to investigate the effectiveness of its model.

We have implemented the parallelization version of EnCal with C and OpenMP [14, 15] on the Sun Enterprise 6000 (Ultra SPARC 168 MHz x 16, 4 Gbyte main memory). Table 1 shows the execution time on Sun Enterprise 6000. Te(4) denotes the 4^{th} degree fragment of relevant logic system T with entailment. The number of conclusions in Te(4) is 10,649. Ee(4) denotes the 4^{th} degree fragment of relevant logic system E with entailment. The number of conclusions in Ee(4) is 15,519. Re(4) denotes the 4^{th} degree fragment of relevant logic system R with entailment. The number of conclusions in Re(4) is 35,027. Table 1 shows that the execution time gets shorter in proportion to the increasement in the number of processors without depending on the number of deduced conclusions. Figure 2 shows the relation between the number of processors and the speed-up ratio against the execution time on 1 processor. Figure 2 shows the same tendency as the theoretical value $2/p$, p is the number of processors, acquired in section 3 was shown.

We have also implemented the parallelization version of EnCal on an 8-node dual

Table 2: The execution time on a clusters of PCs (sec)

Logic systems		1 processor	2 processors	4 processors	8 processors	16 processors
Te(4)	1 processor / 1 node	905	497	272	157	————
	2 processors / 1 node	————	497	273	157	107
Ee(4)	1 processor / 1 node	4706	2413	1285	666	————
	2 processors / 1 node	————	2418	1287	666	406
Re(4)	1 processor / 1 node	31317	15840	8530	4620	————
	2 processors / 1 node	————	15860	8570	4642	2709

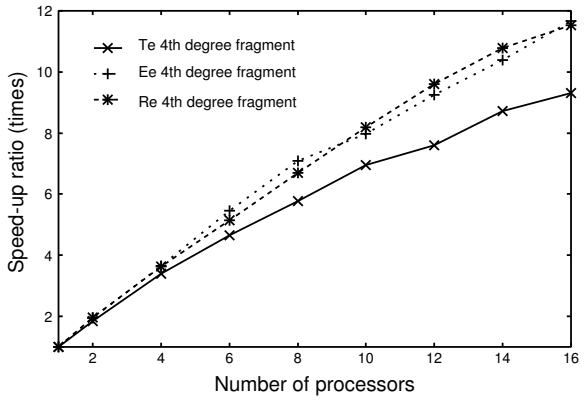

Fig. 3: Speed-up ratio on a clusters of PCs (2 CPU / 1 node)

Pentium III 1GHz PC SMP cluster (i840 chipset, 1GB RDRAM main memory per node, Linux 2.2.16). The nodes on the PC SMP cluster are interconnected through a 100Base-TX Ethernet switch. MPICH-SCore [11] was used as a communication library. We used an intranode MPI library for the PC SMP cluster. All routines were written in C. Table 2 shows the execution time on the cluster of PCs. The column of "1 processor / 1 node" is a case of deducing by 1 processor per 1 node. The column of "2 processors / 1 node" is a case of deducing by 2 processor per 1 node. In Table 2, the execution time gets shorter in proportion to the increasement in the number of processors without depending on the number of deduced conclusions. Figure 3 shows the relation between the number of processors and the speed-up ratio against the execution time on 1 processor, using 2 processors per node. Figure 3 shows the same tendency as the theoretical value $2/p$, p is the number of processors, acquired in Section 3 was shown.

Thus our experiments shows the model of the parallelization version of EnCal is effective for improving the performance of EnCal, independent of the difference in

the architecture and the number of deduced conclusions.

6 Discussion

Improving the performance by parallel processing is effective for not only EnCal but also other forward deduction engines which perform four processes repeatedly; the processes are matching between inference rules and premises, applying inference rules to premises, detecting and reducing the duplicate and adding new conclusions to the set of premises. The model of the parallelization version of EnCal is only designed to reduce the number of times of performing its four processes. It is not considered what an inference rule or data structure EnCal uses, i.e., the model is designed paying attention to the number of times of performing their processes, but not the kinds and natures of their processes. Thus if the inference rule of EnCal change from Modus Ponens into other inference rule, the execution time of parallelization version of EnCal gets shorter in proportion to the increasement in the number of processors. In a forward deduction system for anticipatory reasoning as well as other forward deduction engines whose inference rules and/or data structure are different from EnCal, the execution time of the parallelization version of it therefore gets shorter in proportion to the increasement in the number of processors, too.

7 Concluding remarks

A practical anticipatory system with requirements of high reliability and high security must be able to perform any anticipatory reasoning to get enough effective conclusions anticipatorily during an acceptable time in order to satisfy the requirements from applications. We have presented a model of a parallelization version of EnCal based on master-slave model and implemented it on a shared-memory parallel computer and a clusters of PCs, as a case study to investigate the effectiveness of parallel processing for improving the performance of a forward deduction engine for anticipatory reasoning. Our experiments have shown the the execution time on both architectures gets shorter in proportion to the increasement in the number of processors without depending on the number of deduced conclusions and given premises. Hence, improving the performance by parallel processing is effective for all forward deduction engines for anticipatory reasoning.

References

[1] Anderson Alan R. and Nuel Belnap D. Jr. (1975) *Entailment: The Logic of Relevance and Necessity, Vol. 1.* Princeton University Press.

[2] Anderson Alan R. and Nuel Belnap D. Jr. , and J. Dunn Michael (1992) *Entailment: The Logic of Relevance and Necessity, Vol. 2*. Princeton University Press.

[3] Carriero Nicholas and David Gelernter (1990) *How to Write Parallel Programs: A First Course*. MIT Press.

[4] Cheng Jingde (1996) 'EnCal: An Automated Forward Deduction System for General–Purpose Entailment Calculus'. In: N. Terashima and E. Altman (eds.): *Advanced IT Tools, Proceedings of the 14th WCC, Canberra*. Chapman & Hall, pp. 507–517.

[5] Cheng Jingde (2002) 'Anticipatory Reasoning-Reacting Systems'. In: *Proc. International Conference on Systems,Development and Self-organization (ICSDS'2002)*. Beijing, China, pp. 161–165.

[6] Cheng Jingde (2002), 'Mathematical Knowledge Representation and Reasoning Based on Strong Relevant Logic'. In: R. Trappl (ed.): *Cybernetics and Systems 2002, Proceedings of 16th European Meeting on Cybernetics and Systems Research, Vol. II*. Austrian Society for Cybernetic Studies, pp. 789–794.

[7] Cheng Jingde (2004) 'Temporal Relevant Logic as the Logical Basis of Anticipatory Reasoning-Reacting Systems'. In: D. M. Dubois (ed.): *COMPUTING ANTICIPATORY SYSTEMS: CASYS 2003 - Sixth International Conference*. AIP Conference Proceedings. to appear.

[8] Chirsley Ron (2002) 'Some Foundational Issues Concerning Anticipatory Systems'. *International Journal of Computing Anticipatory Systems* **11**, 3–18.

[9] Dubois Daniel M. (1998) 'Introduction to Computing Anticipatory Systems'. *International Journal of Computing Anticipatory Systems* **2**, 3–14.

[10] Dubois Daniel M (2000) 'Review of Incursive, Hyperincursive and Anticipatory Systems - Foundation of Anticipation in Electromagnetism'. In: D. M. Dubois (ed.): *Computing Anticipatory Systems: CASYS'99 - Third International Conference*. pp. 3–30.

[11] MPICH-SCore., 'PC Clusters Consortium'. http://pdswww.rwcp.or.jp/home-j.html.

[12] Nadin Mihai (2000) 'Anticipation - A Spooky Computation'. *International Journal of Computing Anticipatory Systems* **6**.

[13] Nadin Mihai (2000) 'Anticipatory Computing'. *Ubiquity - The ACM IT Magazine and Forum* **1**. Issue 40.

[14] Omni., 'RWCP OpenMP compiler project'. http://www.hpcc.jp/Omni/.

[15] OpenMP., 'Simple, Portable, Scalable SMP Programming'. http://www.openmp.org/.

[16] Rosen Robert (1985) *Anticipatory Systems - Philosophical, Mathematical and Methodological Foundations*. Pergamon Press.