Parallel Forward Deduction Algorithms of General-Purpose Entailment Calculus on Shared-Memory Parallel Computers

Yuichi Goto, Daisuke Takahashi and Jingde Cheng
Department of Information and Computer Sciences, Saitama University
255 Shimo-Okubo, Saitama-shi, Saitama 338-8570, Japan
E-mail: gotoh@aise.ics.saitama-u.ac.jp
{daisuke, cheng}@ics.saitama-u.ac.jp

Abstract

An automated forward deduction system for entailment calculus is an indispensable component of many application systems – such as theorem finding, active database, knowledge discovery systems – that require an autonomous reasoning engine. The performance of an automated forward deduction system is crucial to its applicability. In this paper, we present some parallel forward deduction algorithms for general-purpose entailment calculus on shared-memory parallel computers. We also present some evaluation results of these algorithms to show their effectiveness and efficiency.

1. Introduction

Reasoning rule generation and automated theorem finding [3, 10, 11] are two fundamental issues in knowledge engineering. To resolve these two problems, it is essential to establish a domain-independent fundamental theory underlying an autonomous reasoning mechanism, and then to develop automatic reasoning tools based on that mechanism.

A decade ago, Cheng proposed some strong relevant logics and showed that an entailment calculus based on those logics can underlie reasoning rule generation in knowledge-based systems and automated theorem finding [3, 4, 5]. We are developing an automated forward deduction system for general-purpose entailment calculus. Named EnCal, this system supports entailment calculus based on strong relevant logics as well as other logics [6].

However, two problems stand in the way of EnCal's applicability as a practical autonomous reasoning engine: slow execution and the large amount of memory needed [7], when EnCal deduces a lot of logical or empirical theorems.

Theorems to be deduced are put on main memory, because EnCal's processing requires frequent access to theorems. So the amount of main memory needed increases linearly, as the number of theorems increases. We are not concerned with this problem so long as the number of theorems is small. However, the problem becomes serious when the number of theorems is large, for example, more than 1 million. It is difficult for sequential computers, such as personal computers (PCs), to solve this problem.

Another problem is the exponential increase in execution as the number of theorems to be deduced increases. So if the number of theorems is more than 1 million, execution will take an impractically long time. Nevertheless, we think that EnCal must be able to deduce large numbers of theorems if it is to become a practical autonomous reasoning engine. Furthermore, the processing speed of sequential computers is becoming saturated. Hence parallel processing is one of the solutions for automated forward deduction systems.

In this paper, we focus on reducing execution rather than saving memory, because parallel computers have quite a bit more main memory than sequential computers like PCs do. We show that execution is reduced by parallelizing EnCal – that is, by adopting two parallel algorithms: one a master-slave model and the other a process-partitioning model. These algorithms are discussed, and the results of implementing these algorithms on shared-memory parallel computers are presented.

2. Terminology

In logic, a notion abstracted from various conditionals is called an "entailment". In general, an entailment – for instance, "A entails B" or "if A then B" must

concern two parts connected by connective " ... entails ...", which are called the antecedent and the consequent of that entailment, respectively. The truth value and/or validity of an entailment depends not only on the truth values of its antecedent and consequent but also, and more essentially, on a necessarily relevant and/or conditional relation between its antecedent and consequent [1, 2].

An entailment calculus is a formal logical system where the notion of entailment is represented by a primitive connective and entailments are a part of its logical theorems.

A formal logic system L is a triplet $(F(L), \vdash_L, Th(L))$ where $F(L)$ is the set of all well-formed formulas of L, $\vdash_L$ is the logical consequence relation of L such that for $P \subseteq F(L)$ and $C \in F(L)$, $P \vdash_L C$ means that within the framework of L taking P as premises, we can obtain C as a valid conclusion, and $Th(L)$ is the set of logical theorems of L such that $\emptyset \vdash_L t$ holds for any $t \in Th(L)$.

For a formal logic system where the notion of entailment is represented by a primitive connective "$\Rightarrow$", 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 formula (also called a first degree entailment) if and only if both A and B are zero degree formulas; a formula of the form $\neg A$ is called a first degree formula if and only if A is a first degree formula; a formula of the form $A * B$, where $*$ is a conjunction or disjunction connective, is called a first degree formula if and only if both A and B are first degree formulas, or either A or B is a first degree formula and the other 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 formula (also called a k^{th} degree entailment) 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 other is a $j^{th}(j < k-1)$ degree formula; a formula of the form $\neg A$ is called a k^{th} degree formula if and only if A is a k^{th} degree formula; a formula of the form $A*B$, where $*$ is a conjunction or disjunction connective, is called a k^{th} degree formula if and only if both A and B are k^{th} degree formulas, or A or B is a k^{th} degree formula and the other is a $j^{th}(j < k)$ degree formula.

Let $(F(L), \vdash_L, Th(L))$ 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 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 $A \in Th^k(L)$, and (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 $Th^k(L)$.

3. EnCal and its processing features

EnCal consists of the following major parts. EnCal-P is a pattern-driven implementation of the inference rule of Modus Ponens (MP for short) for propositional logics. EnCal-Q is an extension of EnCal-P to deal with first order predicate logics. EnCal-E is a tool for reasoning about empirical entailments with a logical theorem schema (LTS for short) generated by EnCal-P and EnCal-Q. EnCal-Q2 is an extension of EnCal-Q to deal with second order predicate logics. EnCal-E2 is an extension of EnCal-E to deal with second order theories. EnCal-T is a tool kit for the user to edit input data for EnCal, transform the theorems into various forms specified by the user, and provide the user with various set operations on the theorems.

This paper focuses on EnCal-P, which is the most basic component of EnCal. EnCal-P works with a pattern pool and a datum pool. For the given axiom schemata of a specified logic L and the degree k, EnCal-P puts all axiom schemata of L in the pattern and datum pools at first, and then repeatedly applies MP to every element of the pattern pool and every element of the datum pool, such that once a new LTS whose degree is not higher than k is reasoned out, it is added to both the pattern and datum pools. This process continues until no new LTS whose degree is not higher than k can be reasoned out.

EnCal-P consists of six parts, as follows.

1. Inputting premises part
2. Pattern matching part
3. Reasoning part; used MP
   (By MP, we can get 'B' from two LTSs, '$A \Rightarrow B$' and 'A'. An LTS deduced by using MP is always the consequent part of an LTS, such as 'B', or an instance of that.)
4. Checking verbose LTSs part
5. Adding a new LTS part
6. Outputting new LTSs part

EnCal-P has a loop structure of from part 2 to part 6, during deducing new LTSs.

In EnCal-P, the heaviest process is Pattern Matching (PM for short). PM judges whether X, which is picked up from the datum pool, is the same as Y, which

is picked up from the pattern pool or an instance of Y. PM is used in two parts, which are the pattern matching part (PM-part for short) and the checking verbose LTSs part (CVL-part for short).

The frequency of PM in these two parts can be made equal to the execution of EnCal-P. To get a complete k^{th} degree fragment of a logic system, we must apply MP to all combinations for already known LTSs in each pool. The frequency of PM in PM-part represents the number of all combinations of LTSs that are produced finally (f-LTSs for short). It is possible to represent the frequency of PM in CVL-part by approximation. When a new LTS is deduced, we must check all already known LTSs to determine whether or not this LTS is verbose.

We try to represent the equations of PM time, which means the time it takes to execute one PM process. The frequency of PM process depend on the number of LTSs to be deduced in each loop. The number of theorems in each loop depend on the number of LTSs which are produced finally and the ratio of success matching in PM-part. Hence these equations are based on two suppositions. First, the number of LTSs to be deduced in each loop depends on normal distribution. Second, the ratio of matching in the PM-part is independent of each loop and is constant.

Following are the parameters:

- the number of LTSs that are produced finally (f-LTS for short) is N_{f};
- the number of premises (axiom schemata of logic) is T;
- the number of loops is l;
- the number of f-LTSs to be deduced in each loop is N_{l_i}, $(1 \leq i \leq l)$;
- the number of verbose LTSs to be deduced in each loop is V_{l_i}, $(1 \leq i \leq l)$;
- the ratio of matching in the PM-part is p;
- A_i, $(1 \leq i \leq l)$ is the divided accumulation probability on normal distribution to l.

Let S_{pm} denote the PM time in the PM-part, giving the following:

$$S_{\mathrm{pm}} = T^2 \cdot N_{\mathrm{f}}^2 + \sum_{i=1}^{l} V_{l_i}. \tag{1}$$

The PM time that depends on N_{f} is independent of N_{l_i}, V_{l_i}. $T^2 \cdot N_{\mathrm{f}}^2$ is the PM time that depends on N_{f}.

N_{l_i} and V_{l_i} depend on p and N_{f}. N_{l_i} is given as follows:

$$N_{l_i} = A_i \cdot N_{\mathrm{f}}. \tag{2}$$

V_{l_i} is the number of LTSs to be matched in the PM-part without N_{l_i}:

$$\begin{aligned} V_{l_i} &= p\left\{N_{l_{i-1}}\left(2T + N_{l_{i-1}} + 2\sum_{k=1}^{i-2} N_{l_k}\right)\right\} \\ &\quad - N_{l_i} \quad (2 < i \leq l), \qquad (3) \\ V_{l_1} &= pT^2 - N_{l_1}, \qquad (4) \\ V_{l_2} &= p\{(T + N_{l_1})N_{l_1} + N_{l_1}T\} - N_{l_2}. \qquad (5) \end{aligned}$$

Let S_{cvl} denote PM time in the CVL-part, giving the following:

$$\begin{aligned} S_{\mathrm{cvl}} &= \sum_{i=1}^{l}\left\{N_{l_i} \cdot \left(T + \sum_{k=1}^{i-1} N_{l_k}\right) + \frac{1}{2}N_{l_i}(N_{l_i} - 1)\right\} \\ &\quad + \frac{1}{2}\left[\sum_{i=1}^{l} V_{l_i} + \sum_{i=1}^{l}\left\{V_{l_i} \cdot \left(T + \sum_{k=1}^{i-1} N_{l_i}\right)\right\}\right]. \end{aligned} \tag{6}$$

In eq. (6), the part of PM time that depends on N_{f} is given as follows:

$$\sum_{i=1}^{l}\left\{N_{l_i} \cdot \left(T + \sum_{k=1}^{i-1} N_{l_k}\right) + \frac{1}{2}N_{l_i}(N_{l_i} - 1)\right\}, \tag{7}$$

and the PM time that depends on V_{l_i} is given as follows:

$$\frac{1}{2}\left[\sum_{i=1}^{l} V_{l_i} + \sum_{i=1}^{l}\left\{V_{l_i} \cdot \left(T + \sum_{k=1}^{i-1} N_{l_i}\right)\right\}\right]. \tag{8}$$

Let S_{total} denote the PM time of two parts, giving the following:

$$S_{\mathrm{total}} = S_{\mathrm{pm}} + S_{\mathrm{cvl}}. \tag{9}$$

S_{total}, S_{pm} and S_{cvl} can be seen similarly in N_{f}, since V_{l_i} and N_{l_i} depend on N_{f}:

$$\begin{aligned} S_{\mathrm{pm}} &\approx 3N_{\mathrm{f}}^2, \qquad (10) \\ S_{\mathrm{cvl}} &\approx 2N_{\mathrm{f}}^2 + N_{\mathrm{f}}^3, \qquad (11) \\ S_{\mathrm{total}} &\approx 5N_{\mathrm{f}}^2 + N_{\mathrm{f}}^3. \qquad (12) \end{aligned}$$

S_{cvl} is N_{f} times as large as S_{pm}. The processing PM in the CVL-part becomes heavier as the number of f-LTSs increases.

Therefore, in order to increase the PM processing number in unit time, we need to distribute the PM processes among more than two processors.

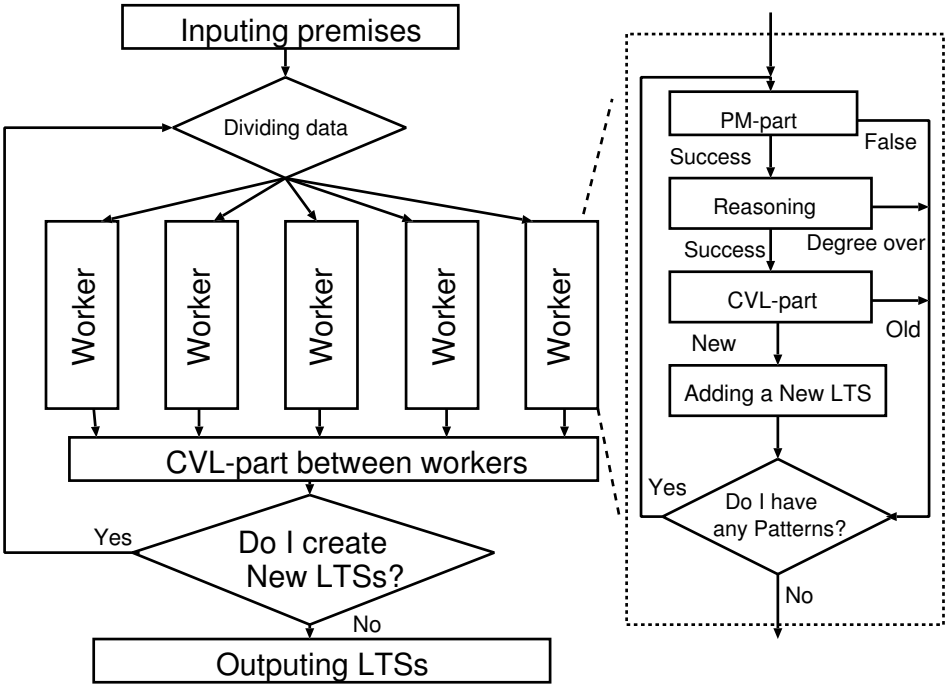

Figure 1. Master-slave model

4. Two parallel algorithms

We now present two parallel forward deduction algorithms of EnCal-P on shared-memory parallel computers.

4.1. Master-slave model

A parallel algorithm of EnCal-P on shared-memory parallel computers is based on the master-slave model. This model is shown in Fig. 1. The equation of PM time in this model consists of three parts: the PM-part, the CVL-part, and the between-workers CVL-part. Let P denote the number of processors. Let $M_{\rm pm}$, $M_{\rm cvl}$ denote PM times of PM-part and CVL-part, which are represented as follows, for use in $S_{\rm pm}$ or $S_{\rm cvl}$:

$$M_{\rm pm} = \frac{1}{P} S_{\rm pm}, \tag{13}$$

$$M_{\rm cvl} = \frac{1}{P} S_{\rm cvl}. \tag{14}$$

In the between-workers CVL-part, the reasoning result of one worker may include verbose LTSs for the results of the other worker. In loop number i, the number of verbose LTSs of one worker is at most $\sum_{k=1}^{i-1}(N_{l_k}) - \frac{1}{P}(N_{l_i} + V_{l_i})$. So let $M_{\rm worker}$ denote PM time in the between-workers CVL-part, as shown in the following equation:

$$\begin{aligned} M_{\rm worker} &= \frac{P-1}{2P}\left(\sum_{i=1}^{l} N_{l_i}^2\right) \\ &\quad + \frac{P-1}{4}\sum_{i=1}^{l}\left\{N_{l_i}\left(\sum_{k=1}^{i-1} N_{l_k} - \frac{1}{P}N_{l_i}\right)\right\} \\ &\quad + \frac{Pl}{4}(l+1). \end{aligned} \tag{15}$$

Let $M_{\rm total}$ denote PM time in this model:

$$M_{\rm total} = \frac{1}{P} S_{\rm total} + M_{\rm worker}. \tag{16}$$

$M_{\rm total}$ can be approximated by $N_{\rm f}$:

$$M_{\rm worker} \approx P \cdot N_{\rm f}^2, \tag{17}$$

$$M_{\rm total} \approx \frac{1}{P}(5N_{\rm f}^2 + N_{\rm f}^3) + P \cdot N_{\rm f}^2. \tag{18}$$

This model has the following advantages.

1. If the number of f-LTSs ($N_{\rm f}$) is $P^2 \ll N_{\rm f}$ (from eq. (18)), the execution spent on all PM shrinks in proportion to the number of processors.

2. Therefore, we know two things:

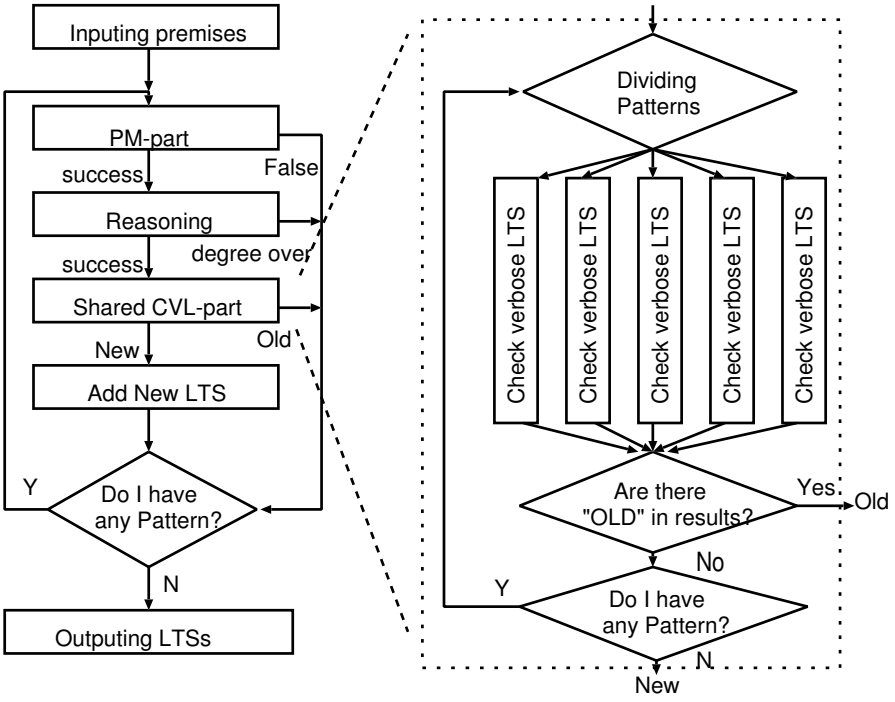

Figure 2. Process-partitioning model

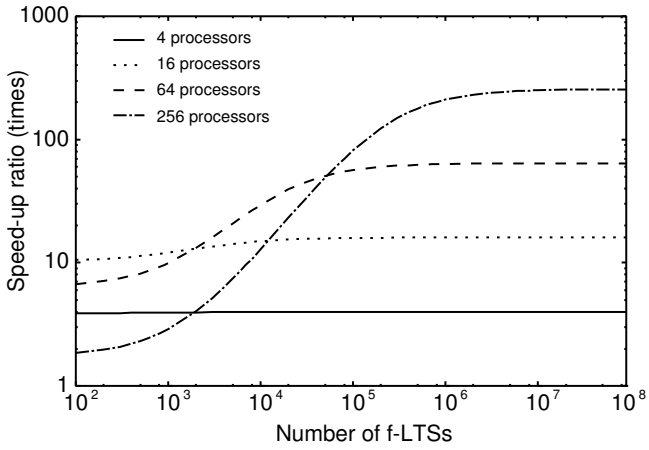

Figure 3. Theoretical speed-up ratio (Master-slave model)

(a) If there are fewer than 10 processors, the execution to be spent on all PM shrinks in proportion to the number of processors without N_{f}.

(b) If there are more than 10 processors, we can expect to shrink the execution in proportion to the number of processors when N_{f} is more than P^3.

Fig. 3 shows the theoretical speed-up ratio and the number of f-LTSs.

This model has a disadvantage in that there may be many instances of f-LTSs. In this paper, 'instance' means the following. By MP, we can get '$A \Rightarrow C$' from two LTSs, '$(A \Rightarrow B) \Rightarrow (A \Rightarrow C)$' and '$A \Rightarrow B$'. '$A \Rightarrow C$' is a logical formula, the same as '$A \Rightarrow B$'. Therefore, '$A \Rightarrow C$' is an instance of '$A \Rightarrow B$'. For example, let n denote that the number of instances is n times as large as the number of core f-LTSs, which includes no instances. Let $N_{\mathrm{f}}^{'}$ denote the number of core-fLTSs, and M_{total} including instances is given as follows by $N_{\mathrm{f}}^{'}, n$:

$$\begin{aligned} M_{\mathrm{total}} &\approx \frac{1}{P}(5N_{\mathrm{f}}^2 + N_{\mathrm{f}}^3) + P \cdot N_{\mathrm{f}}^2 \\ &\approx \frac{(n+1)^2}{P}(5N_{\mathrm{f}}^{'2} + N_{\mathrm{f}}^{'3}) + (n+1)^2 P \cdot N_{\mathrm{f}}^{'2}. \end{aligned} \tag{19}$$

If P is more than n or n is constant, we expect to decrease the execution in proportion to the number of processors.

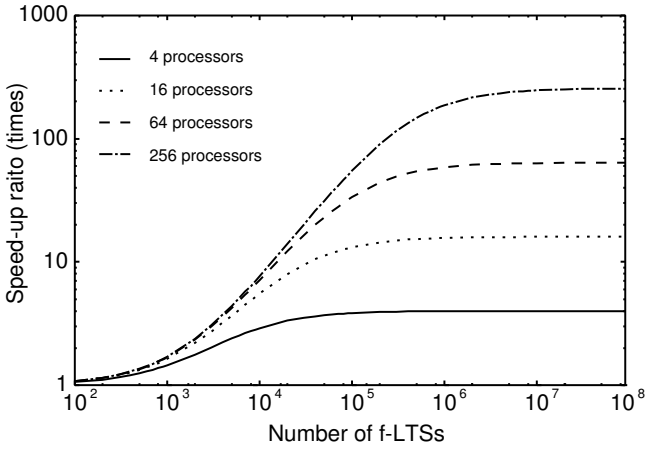

Figure 4. Theoretical speed-up ratio (Process-partitioning model)

4.2. Process-partitioning model

Another parallel algorithm is based on the process-partitioning model. This model is shown in Fig. 2.

The main feature of this model is that only PM processes in the CVL-part are shared among processors. Other parts are processed by one processor. Let $P_{\rm pm}$, $P_{\rm cvl}$, and $P_{\rm total}$ denote the PM time of the PM-part, the CVL-part, and both parts by $S_{\rm pm}, S_{\rm cvl}$:

$$P_{\rm pm} = S_{\rm pm}, \tag{20}$$

$$P_{\rm cvl} = \frac{1}{P} S_{\rm cvl}, \tag{21}$$

$$P_{\rm total} = S_{\rm pm} + \frac{1}{P} S_{\rm cvl}. \tag{22}$$

As follows can be approximated by $N_{\rm f}$:

$$P_{\rm total} \approx 3N_{\rm f}^2 + \frac{1}{P}(2N_{\rm f}^2 + N_{\rm f}^3). \tag{23}$$

This model has the following advantages:

1. If $N_{\rm f}/P$ is more than 10^4, the execution to be spent on all PM shrinks in proportion to the number of processors.
2. The number of instances included in f-LTSs is as large as the result of the sequential model.

Fig. 4 shows a theoretical speed-up ratio and number of f-LTSs.

The disadvantage of this model is that if $N_{\rm f}/P$ is less than 10^4, we cannot expect the execution spent on all PM to shrink in proportion to the number of processors, because $P_{\rm pm} \ll P_{\rm cvl}$ is not satisfied when $N_{\rm f}$ is small.

5. Implementation and results

We have implemented only the master-slave model with C and Open MP [8, 9] on the Sun Enterprise 6000 (Ultra SPARC 168 MHz x 16, 4 Gbyte main memory).

Table 1 shows the execution data of the sequential model and the master-slave model. Ten(3) denotes the 3rd degree fragment of the relevant logic system T with entailment and negation. CMLen(3) denotes the 3rd degree fragment of classical mathematical logic with entailment and negation. Te(4) denotes the 4th degree fragment of relevant logic system T with entailment. Ee(4) denotes the 4th degree fragment of relevant logic system E with entailment. Re(4) denotes the 4th degree fragment of relevant logic system R with entailment.

Table 1 shows us several things:

1. The number of core f-LTSs to be based on one logic theory is a value different from the same degree fragment to be based on another.
2. The execution depends on the number of f-LTSs and the complexity of the logic theory.
3. The included instance ratio in the f-LTSs depends on the number of processors.
4. The execution shrinks in proportion to the number of processors.

1. is a reason why it is difficult for us to predict the execution of EnCal. We cannot know it without executing EnCal.

2. is evidence of which we represented the equations. The number of f-LTSs and the complexity of logic theories influence the frequency of PM processing. So the execution also increases when the number of f-LTSs increases or when the number of f-LTSs is deduced from more complicated theorems.

We discuss 3. and 4. with the experimental results and equations. Fig. 5 shows the including instances in the f-LTSs ratio and the number of processors. The including instance ratio becomes constant when the number of processors increases. The number of instances in f-LTSs to be deduced from CMLen, which is the highest ratio among these 16 processors, is only 2.7 times as large as the number of core f-LTSs. If the including instance ratio is constant with a large number of processors, $(n+1)^2 < P$ (from eq. (19)) must be satisfied. Therefore, we can expect that the speed-up ratio increases with the number of processors.

Fig. 6 shows the speed-up ratio of observed values. In the result of Ten(3), the speed-up ratio is saturated for the number of processors. The reason for this is that

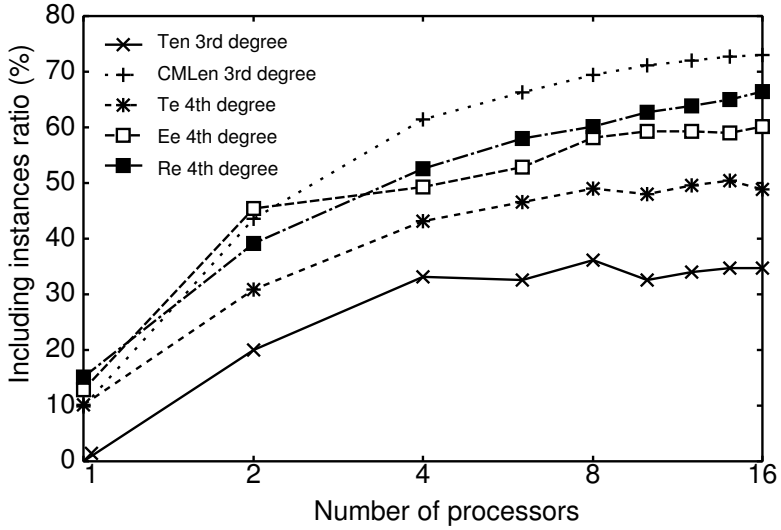

Figure 5. Including the instance ratio in f-LTSs

Table 1. Real execution time

Logic systems (k^{th} degree)	sequential (core f-LTS/f-LTSs)	2 processors (core f-LTS/f-LTSs)	4 processors (core f-LTS/f-LTSs)	8 processors (core f-LTS/f-LTSs)	16 processors (core f-LTS/f-LTSs)
Ten(3)	0.617s (252/252)	0.397s (252/315)	0.252s (252/377)	0.237s (252/395)	0.211s (252/386)
CMLen(3)	3h 31m 22s (10649/11808)	2h 5m 44s (10649/18876)	1h 8m 52s (10649/27569)	37m 12s (10649/34899)	17m 38s (10649/39543)
Te(4)	46m 45s (10046/11184)	24m 43s (10046/14518)	14m 2s (10046/17681)	8m 14s (10046/19698)	5m 23s (10046/19667)
Ee(4)	2h 42m 55s (15519/17802)	1h 37m 28s (15519/28461)	50m 54s (15519/30583)	30m 26s (15519/37125)	19m 15s (15519/38957)
Re(4)	20h 3m 51s (35027/41280)	11h 3m 26s (35027/57569)	5h 54m 55s (35027/73926)	3h 15m 39s (35027/87910)	2h 11m 43s (35027/104461)

$P^2 \ll N_{\mathrm{f}}$ (from eq. (18)) is not satisfied. Other results show that the speed-up ratio increases with the number of processors. We can conclude that the theoretical expressions are proper and that the master-slave model is effective.

Following are the reasons why observed values differ from the theoretical values.

1. The overhead of thread generation influences the execution.

2. The number of processors is small although the including instance ratio increases.

6. Concluding remarks

In this paper, we have presented two parallel forward deduction algorithms of EnCal-P on shared-memory parallel computers.

The process-partitioning model is not effective enough to be implemented on shared-memory parallel computers because we cannot expect to efficiently reduce the execution when the number of f-LTSs is less than 10^4.

The master-slave model, on the other hand, is effective enough to be implemented on shared-memory parallel computers. The execution is decreased regardless of the number of f-LTSs. This conclusion was confirmed by the results of implementation.

Many important and challenging research problems

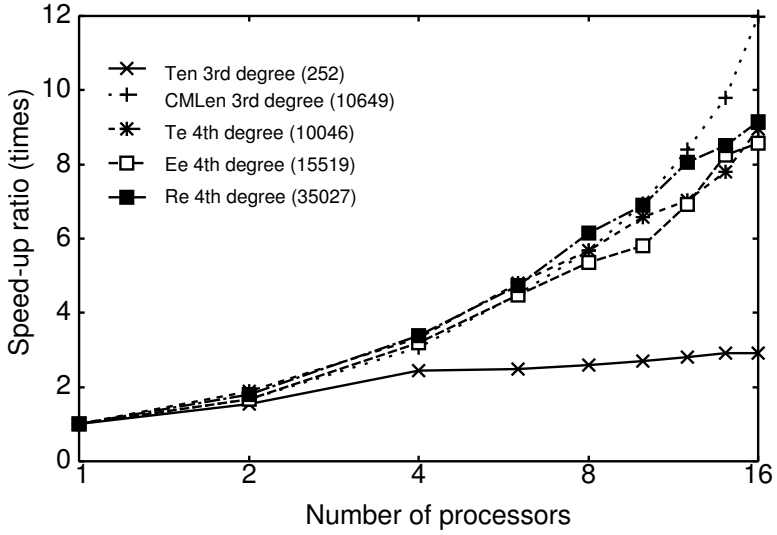

Figure 6. Practical speed-up ratio

lie in this direction. For example, the master-slave model has a disadvantage, which is that it includes instances in f-LTSs. If we wish to improve the master-slave model, we have to decrease the number of instances in f-LTSs.

In this paper, we have not considered saving memory. If we need to deduce 10^6 theorems, we think that this implementation may use 1.6 Gbyte of main memory. For the sake of practical applications, it is important to conserve memory.

When EnCal deduces more than 10^6 LTSs, we must implement it on a distributed-memory parallel computer. So, to investigate those algorithms, we will have to design data structures and algorithms on distributed-memory parallel computers.

References

[1] A. R. Anderson and N. D. Belnap Jr. *"Entailment: The Logic of Relevance and Necessity"*, vol. 1, Princeton University Press., 1975.

[2] A. R. Anderson, N. D. Belnap Jr., and J. M. Dunn, *"Entailment: The Logic of Relevance and Necessity"*, vol. 2, Princeton University Press., 1992.

[3] J. Cheng, "Entailment Calculus as a Logical Tool for Reasoning Rule Generation and Verification," in J. Liebowitz (Ed.), *"Moving Toward Expert Systems Globally in the 21st Century,"* pp. 386–392, Cognizant Communication Co., 1994.

[4] J. Cheng, "Entailment Calculus as the Logical Basis of Automated Theorem Finding in Scientific Discovery," in *"Systematic Methods of Scientific Discovery – Papers from the 1995 Spring Symposium,"* AAAI Technical Report SS-95-03, pp. 105–110, 1995.

[5] J. Cheng, "The Fundamental Role of Entailment in Knowledge Representation and Reasoning", *Journal of Computing and Information*, vol. 2, no. 1, pp. 853–873, 1996.

[6] J. Cheng, "Encal: An Automated Forward Deduction System for General–Purpose Entailment Calculus", In *N. Terashima and E. Altman (Eds.) "Advanced IT Tools" IFIP World Conference on Advanced IT Tools IFIP 96 – 14th World Computer Congress*, pp. 507–517, 1996.

[7] K. Nishi, J. Cheng, K. Ushijima, "Improving the Performance of Automated Forward Deduction System EnCal", *Proc. International Symposium on High Performance Computing (ISHPC97)*, Nov. 1997, Lecture Notes in Computer Science, vol. 1336, pp. 371–380, Springer-Verlag (1997).

[8] Omni. RWCP OpenMP compiler project. http://pdplab.trc.rwcp.or.jp/pdperf/Omni/.

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

[10] L. Wos, *Automated Reasoning: 33 Basic Research Problems*, Prentice-Hall., 1988.

[11] L. Wos. "The Problem of Automated Theorem Finding", *Journal of Automated Reasoning*, vol. 10, no. 1, pp. 137–138, 1993.