ICOT Technical Report: TR-0868

TR-0868

モデル生成型定理証明器 MGTP の並列化方式

長谷川　隆三、越村　三幸

April, 1994

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

モデル生成型定理証明器 MGTP の並列化方式

長谷川 隆三　越村 三幸
ICOT

概要

MGTP は、並列論理型言語 KL1 を用いて並列推論マシン PIM 上に作成されたモデル生成法に基づく一階述語論理定理証明器である。本稿では、非ホーン節に対する場合分けによって生じる OR 並列性及び、節の前件リテラルとモデル候補アトムとの連言照合操作に含まれる AND 並列性を抽出する技法と KL1 による効率的な実装法について述べる。ここで提示した OR 並列化方式及び AND 並列化方式は共に、幾つかのベンチマーク問題に対して、PIM/m 256 PE 上で線形に近い良好な台数効果を達成している。

1　まえがき

一階述語論理の定理証明は第五世代コンピュータの基本概念である「論理に基づく推論」と密接な関係があり、Prolog や KL1 などの論理型言語を生み出す源となった重要な基礎技術である。一階述語論理は論理型言語を基礎づけるホーン論理を含んでおり、記述力の点では優れているが、その定理証明系は、実行効率が悪く、小さな問題でも非常に大きな計算資源を必要とするため、実用性の観点からこれまであまり顧り見られなかった。しかし、近年の論理プログラミング技術の進展を背景に、これを見直す動きが出始めている。その典型例は Prolog 技術をうまく使った PTTP[1] や SATCHMO[2] などである。

筆者らは、論理プログラミング技術と自動推論技術の融合を目指し、平成 2 年より一階述語論理のモデル生成型定理証明系 MGTP の研究を開始した [3][4][5]。本研究の目標は、並列論理型言語 KL1 の特長を活用して高速な並列自動推論システムを並列推論マシン PIM 上に構築することである。定理証明の過程には多大な並列性が内在しており、膨大な計算量、大量の記憶域を必要とするので、定理証明系自身、KL1 及び PIM にとって格好の応用の一つでもある。

定理証明方式として、SATCHMO のモデル生成法を採用した。その理由は、以下の通りである。

1) モデル生成法は、基底 (グランド) アトムのみがモデル要素として生成される場合、ユニフィケーションが不要でマッチングで十分であるので、KL1 が提供する高速なマッチング (ヘッド・ユニフィケーション) を利用して、効率良く実装できる。

2) 数学的定理のように深い推論 (長い証明) が必要となる問題を解く場合、モデル生成法のようなボトムアップ型の定理証明器は、探索空間を狭めるための各種の技法、すなわち補題化、包摂テスト、削除戦略等を容易に組み込むことができる。

現在、変数を含まない基底アトムのみを扱うグランド版と、変数含みのアトムを扱うノングランド版の、2 種の MGTP が開発されている。

グランド版 MGTP の場合、ユニフィケーションが不要でマッチングですむので KL1 との親和性が良く、簡潔かつ効率の良い証明器が得られた。また、グランド版は、ノンホーン節のケース分割によって OR 並列性が容易に抽出できるという利点を持っている。

一方、ノングランド版 MGTP の場合、オカーチェック付きユニフィケーションをユーザが (KL1 で) 書く必要があり、1) で述べた KL1 の利点を活用できない。したがって、グランド版ほどの高速性は期待できないが、KL1 によるユニファイアの実装は、KL1 の実用性をみる上で良い評価材料となった (C で書かれたものと比べて、2～3 倍程度の遅さという良好な結果が得られている)。また、ノングランド版では、ケース分割が生じない (OR 並列性のない) ホーン節に対象を限定しているので、これから如何に (AND) 並列性を抽出するかという、AND 並列化方式の確立が研究の主眼であった。

現在の所、両版とも PIM/m-256PE 上で 200 倍以上の台数効果を達成している。グランド版 MGTP を用いて、有限代数の問題を試みた結果、カナダの数学者 Bennett により提示された準群に関する未解決問題の一部を解くことに成功した [6][7][8]。また、ノングランド版 MGTP は、

PIM/m の単一 PE 上で、米国アルゴンヌ国立研究所 (ANL) 開発の OTTER と同程度かあるいはこれをしのぐ性能を達成しており、OTTER では解けなかったハードな数学的定理を解くことにも成功している [9]。

MGTP の研究過程で、SATCHMO に含まれていた連言照合の冗長性を除去する手法、ケース分割の爆発を防ぐ手法 [10]、アトムの過剰生成を抑制して計算量・記憶域を削減する手法 [11]、などのモデル生成アルゴリズムの改良技術や、論理プログラミングで開発された「失敗による否定」を組み込む技術 [12]、さらに様相論理システムや仮説推論システムなどの構築技術 [13][14] が生み出されてきた。

本稿では、これらの技術を支える上で最も重要な課題であり、我々がこの 1、2 年精力を注いできた MGTP の並列化方式とその実現手法に焦点をあてる。以下では、まずモデル生成法とは何かを概説し、MGTP の証明手続き及び基本的な処理構造について簡単に述べる。そして MGTP の証明過程における並列性の所在について言及し、OR 並列化方式及び AND 並列化方式の考え方と実現方法を紹介する。最後に実験結果と今後の課題について述べる。

2 モデル生成法

本文を通じて節は次のように含意式の形で表現される。

$$A_1, A_2, \ldots, A_n \rightarrow C_1; C_2; \ldots; C_m$$

ここで、$A_i (1 \leq i \leq n; n \geq 0)$ および $C_j (1 \leq j \leq m; m \geq 0)$ はアトムである。$\rightarrow$ の左側を節の前件、右側を後件という。前件は $A_1, A_2, \ldots, A_n$ の連言 (',')、後件は $C_1, C_2, \ldots, C_m$ の選言 (';') である。前件が $true$ ($n = 0$ のとき) の節 $true \rightarrow C_1; C_2; \ldots; C_m$ を正節、後件が $false$ ($m = 0$ のとき) の節 $A_1, A_2, \ldots, A_n \rightarrow false$ を負節、それ以外の節 ($m \neq 0, n \neq 0$) は混合節と呼ばれる。さらに、正節及び混合節を generator 節、負節を tester 節とも呼ぶ。

モデル生成法には次の二つの規則がある。

- モデル拡張規則: 混合節もしくは正節において、ある置換 σ のもとに、各前件アトム $A_i\sigma$ があるモデル候補 M で充足されており、いずれの後件アトム $C_j\sigma$ も M で充足されてないとき、各 $C_j\sigma$ を M に加えてモデル候補を拡張する。
- モデル棄却規則: 負節 において、ある置換 σ のもとに、各前件アトム $A_i\sigma$ があるモデル候補 M で充足されるとき、M を棄却する。

ここで、$(A_1, A_2, \ldots, A_n)\sigma$ を得る過程を節の前件とモデル候補要素との連言照合という。正節の前件 ($true$) はどんなモデルでも充足されることに注意されたい。

M を拡張する際に $C_j\sigma$ が M で充足されてるか否かを検査しているが、これは包摂テストと呼ばれる検査の一種である。二つの項 u, v 間に $u\sigma = v$ が成り立つとき、u は v を包摂するといい、このように、より一般的な項を残す操作を包摂テストという。モデル拡張によって生成されたアトムがモデル候補要素に包摂されるか否かを検査することを前向き包摂テスト、逆にモデル候補要素を包摂するようなアトムかどうかを検査することを後向き包摂テストという。

モデル生成法は、上記の二つの規則に従って、与えられた節集合に対するモデル[†1]を構成的に求めていくものである。先ず、初期モデル候補集合として $\mathcal{M} = \{\phi\}$ から始め、いずれの規則も適用することができなくなった時点で、最終的なモデル候補の集合 $\mathcal{M}$ を出力する。このとき、すべての $M \in \mathcal{M}$ について、M はすべての節を満足しているため、与えられた節集合のモデルとなる。もし、$\mathcal{M} = \phi$ であれば、すべてのモデル候補が棄却されておりモデルが存在しないため、その節集合は矛盾している (あるいは充足不能である) ことを示す。

例として次の節集合 S1 を考える。

$$C1: \quad p(X), s(X) \rightarrow false.$$
$$C2: \quad q(X), s(Y) \rightarrow false.$$
$$C3: \quad q(X) \rightarrow s(f(X)).$$
$$C4: \quad r(X) \rightarrow s(X).$$
$$C5: \quad p(X) \rightarrow q(X); r(X).$$
$$C6: \quad true \rightarrow p(a); q(b).$$

S1 問題に対する証明木を図 2.1に示す。まず、初期モデル候補集合として $M_0 = \{\phi\}$ から始める。$C6$ にモデル拡張規則を適用することにより、M_0 は $M_1 = \{p(a)\}$ と $M_2 = \{q(b)\}$ に場合分けされる。次に、$C5$ によって、M_1 は $M_3 = \{p(a), q(a)\}$ と $M_4 = \{p(a), r(a)\}$ に場合分けされる。さらに、$C3$ によって M_3 は $M_5 = \{p(a), q(a), s(f(a))\}$ に拡張される。さて、M_5 においてモデル棄却規則が $C2$ に適用できて、M_5 は棄却され、このケースの処理はここで終了する。一方、M_4 は $C4$ によって $M_6 = \{p(a), r(a), s(a)\}$ に拡張されるが、$C1$ によって棄却される。同様に、M_2 は $C3$ によって $M_7 =$

†1 ここでモデルとは、真であると解釈されるアトムの集合のことである。モデルに含まれないアトムは、偽であると解釈される。

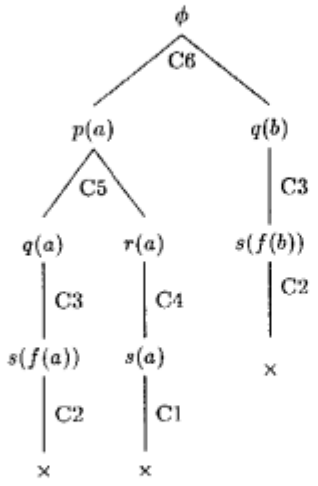

図 2.1 S1 に対する証明図

$\{q(b), s(f(b))\}$ に拡張された後、$C2$ によって棄却される。今や、モデルを構成する可能性のすべてが絶たれたので、S1 は充足不能であると結論できる。

3 MGTP の基本構造

3.1 モデル生成アルゴリズム

MGTP で採用しているモデル生成アルゴリズムを図 3.1 に示す。但し、本図は証明木の一つの枝の探索を行う手続きを示したものであり、場合分けに対する手続きは含まれていない。ここで、M はモデル候補を、D はモデル拡張候補 (モデル拡張規則の適用の結果として M に付加されるべきアトムの集合) を、Δ は D の要素をそれぞれ表す。M と D の初期値はそれぞれ空集合および正節の後件アトムの集合である。

図 3.1 において、(4),(6),(8) がモデル拡張規則、(7) がモデル棄却規則に対応している。**while** ループを一回りする間に、(3) D から Δ を一つ選び、(4) Δ と M を使って (混合節 *generator* による連言照合を行なうことにより) 新たなモデル拡張候補アトムの集合 *new* を生成する。次に、(6) *new* の包摂テストを $M \cup D$ に対して行ない、包摂されなかったアトム集合を *new'* とする。そして、(7) *new'* と $M \cup D$ を使って棄却テスト (負節 *tester* による連言照合) を行ない、成功すれば手続きは終了する。さもなければ (8) D を *new'* で拡張する。このサイクルの始めに D が空ならば反駁に失敗、即ちモデル M が見つかったことになり、このアルゴリズムは停止する。

図 3.1に示した連言照合 CJM は、以下のような集合上の関数として定義される。

$$
\begin{aligned}
&CJM_{Cs}(X,Y) = \\
&\quad \{\sigma C \mid A_1,\ldots,A_n \rightarrow C \in Cs \\
&\qquad \wedge\ \sigma A_1,\ldots,\sigma A_n \rightarrow \sigma C \\
&\qquad \wedge\ \forall i(1 \le i \le n)\exists B(B \in X+Y)\sigma A_i = \sigma B\}
\end{aligned}
$$

ここで、Cs は節集合、Y は既に連言照合済みのアトム集合、X はこれから連言照合をしようとする新たなアトムまたはアトム集合を表す。$X+Y$ は、X がアトムの時は $\{X\} \cup Y$ を、X がアトム集合の時は $X \cup Y$ を表す。

3.2 連言照合における冗長性の除去

モデル生成法にとって連言照合は基本的操作であるので、どのような操作を具体的に行なうのかを少し詳しく述べることにする。n 個の前件を持つ節 $A_1, A_2, \ldots, A_n \rightarrow \ldots$ の連言照合をモデル候補 M に対して行なう場合、M の要素の n 個の組 $B_1, B_2, \ldots, B_n$ 全てについて B_i と A_i $(i = 1,\ldots,n)$ との照合を行なう必要があるが、連言照合における冗長性の回避は重要である。

二つの前件アトムを持つ節 $A_1, A_2 \rightarrow C$ を考えよう。この節に対する連言照合を行なうためには、モデル候補 M の要素の対を全て試みる必要がある。そして全ての要素対に対する連言照合の結果、新たなモデル拡張候補が得られ D に追加されたとしよう。次の段階では D から要素 Δ を一つ選び、$M \cup \{\Delta\}$ からの要素対を試みることになる。この対の数は全部で

$$(M \cup \{\Delta\})^2 = M \times M \cup M \times \Delta \cup \Delta \times M \cup \Delta \times \Delta.$$

である。

しかし $M \times M$ 個の対については、前の段階で既に試みられているので、これらの対に対する連言照合は冗長になる。これを防ぐには、要素 Δ を少なくとも一つ持つ対のみ、即ち $M \times \Delta, \Delta \times M, \Delta \times \Delta$ を試せばよい。前件アトム数が 3 以上の節の場合にも同様のことがいえる。

図 3.2 に節 $A_1, A_2 \rightarrow C$ に対する連言照合の過程をマトリックスを用いて示す。モデル拡張候補 D の初期値として与えられる正節の後件アトム、及び連言照合の結果生成されたアトムには、1 から順に番号がふられる。このマトリックスの i 行 j 列の要素は、A_1 に i 番目、A_2 に j 番目のアトムを照合させることを意味し、(i,j) 要素対に対する連言照合を $i \times j$ と書く。

初期値として、$M = \phi$, $D = \{1\}$ が与えられたとしよう。そして、照合は全て成功し、結果のアトムが生成さ

(0) $M := \phi$;
(1) $D := \{A \mid (true \rightarrow A) \in$ a set of given clauses$\}$;
(2) **while** $D \neq \phi$ **do begin**
(3) $\quad D := D \setminus \{\Delta\}$;
(4) $\quad new := CJM_{generator}(\Delta, M)$;
(5) $\quad M := M \cup \{\Delta\}$;
(6) $\quad new' := subsumption(new, M \cup D)$;
(7) $\quad$ **if** $CJM_{tester}(new', M \cup D) \ni false$ **then return**(unsat);
(8) $\quad D := D \cup new'$;
(9) **end return**(sat)

図 3.1 MGTP のアルゴリズム

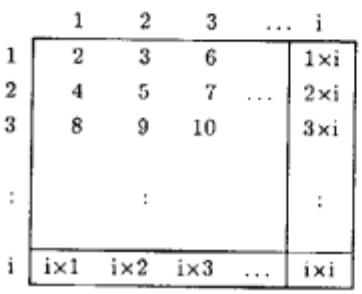

図 3.2 連言照合操作

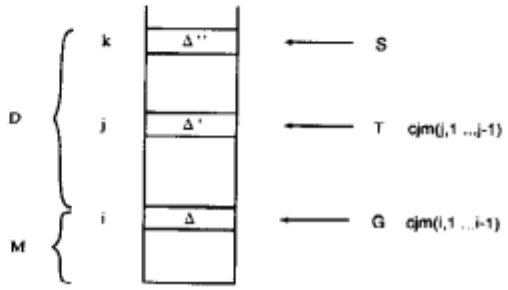

図 3.3 MGTP の実装

れるものと仮定する。この場合、先ず $\Delta = 1$ として、Δ どうしの掛け合わせ (1×1) のみが行われ、その結果に 2 がふられる。そして、使用した Δ は M に、新たに生成されたアトムは D に追加され、$M = \{1\}$, $D = \{2\}$ となる。次に、$\Delta = 2$ として、Δ と M を用いた掛け合わせが行われ、1×2 の結果に 3、2×1 の結果に 4、2×2 の結果に 5 がふられ、$M = \{1,2\}$, $D = \{3,4,5\}$ となる。同様に $\Delta = 3$ として、連言照合操作が続けられる。

以下では、i 番目のアトムに対するモデル拡張 $CJM_{generator}(i, M)$ とは、M の全要素 $1, \ldots, i-1$ に対して $(1 \ldots i-1) \times i$, $i \times (1 \ldots i-1)$, $i \times i$ の連言照合を行い、モデル候補となるべき新たなアトムを生成することを指す。生成されたアトムについては、その生成番号順に包摂テストが行なわれる点に注意されたい。

3.3 MGTP の処理構造

MGTP のモデル生成アルゴリズムの主な操作は以下の三つである。

(1) 混合節 *generator* による連言照合を行ない、アトム集合を生成する。

(2) 生成されたアトム集合の包摂テストを行ない、包摂されないアトムをモデル拡張候補に追加する。

(3) 負節 *tester* の連言照合により、モデル候補の棄却テストを行なう。

MGTP は、これら三つの操作に対応して三種類のプロセスから構成されている。それらのプロセスをそれぞれ、生成 (G) プロセス、包摂テスト (S) プロセス、棄却テスト (T) プロセス と呼ぶ。

生成されたアトムは、生成順を保つため、リストの形で蓄えられる。G,S,T の各プロセスは、リストの要素へのポインタ i, j, k を保持し、以下の手続きを繰り返す (図 3.3)。

G: i 番目の要素 (Δ) と $1 \ldots i-1$ 番目の各要素 (M) を用いて連言照合 $(CJM_{generator}(i, 1 \ldots i-1)$ によるモデル拡張) を行い、この結果生成されたアトムを S プロセスに送る。生成アトムを全て送った後、$i := i+1$ とする。

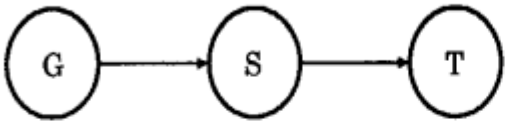

図 3.4 G、S、T 間のデータの流れ

S: G から受けとったアトム Δ'' の包摂テストを $M \cup D$ の各要素 $(1 \ldots k)$ に対して行う。Δ'' が包摂された場合これを放棄し、包摂されない場合は Δ'' を $k+1$ 番目の要素として登録し、$k := k+1$ とする。

T: j 番目の要素 (Δ') と $1 \ldots j-1$ 番目の各要素を用いて連言照合 $(CJM_{tester}(j, 1 \ldots j-1)$ による棄却テスト) を行う。これが成功した場合、つまり、モデル が棄却された場合、他の全てのプロセスに中断指令を送り終了する。失敗した場合には、$j := j+1$ とする。

この手続きを並列化する際には、複数の G,S,T プロセスが生成されるので、それぞれが保持している複数のポインタ i, j, k の排他制御が必要になる。

G,S,T の関係は生成されたアトム (データ) の流れに着目すると、図 3.4 のようになる。この図より容易に想像できるように、MGTP の計算機構は G プロセスを generator、T プロセスを tester とした generate-and-test 型になっている。一般に generate-and-test の計算では、generator の走り過ぎによるアトムの過剰生成の危険性がある。これは tester にかける必要のない無用アトムを生成してしまい[†1]、それらに対して高価な包摂テストが無駄に行われることを意味し、メモリ空間の爆発を引き起こすことにもなる。特に並列環境下においては、generator に対する制御が何らなされない場合は、generator が暴走し、いっこうに tester が実行されないという事態も許しかねない。この危険を回避するため我々は、「tester が必要とする時のみ generator を起動しアトムを生成する」という要求駆動的考え方に基づいた、遅延モデル生成法と呼ぶ新たな方法を導入した [11]。

図 3.5に、遅延モデル生成法のアルゴリズムを示す。本アルゴリズムでは、主に G プロセスと T プロセスの二つが交信しながら独立並行に動作する。T プロセスは、1) G プロセスにモデル拡張候補アトム Δ を要求し、2) これまでのテスト済みのアトム集合 $(M \cup D)$ に対して Δ の包摂テストを行った後、3) 棄却テストを行う。一方 G プロセスは、1) 生成したアトムを蓄える Buf が空のとき、これを埋める準備をするため、D から一個のアトム e を選んで Buf にモデル拡張コードを設定しておく (**delay** $CJM_{generator}(e, M)$)。そして、2) T プロセスからの Δ の要求を待って、3) Buf に起動をかけ (**force** Buf)、モデル拡張候補アトムを生成する。

†1 ある深さで **false** を導くような反駁アトムが見つかるものとすると、それ以上の深さのアトムまで生成してしまうこと

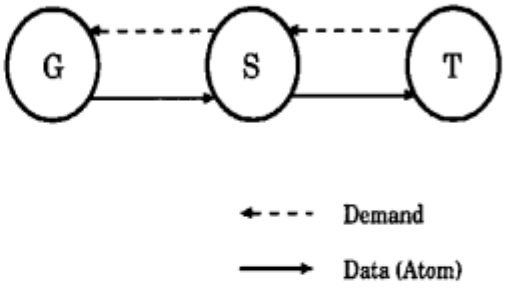

図 3.6 G、S、T 間のデータの流れ

遅延モデル生成法では、データの流れとは逆向きに要求が流れる (図 3.6)。G プロセスは S プロセスからの、S プロセスは T プロセスからの要求があってから動き始める。例えば、G プロセスは S プロセスからの要求個数分だけアトムを生成すると、S プロセスからの要求待ち状態になる。

このような要求駆動制御により、generator と tester のスピード差が均一化され、生成の行き過ぎによる無駄をなくすことができ、計算量及びメモリ消費量のオーダを大幅に削減することができる。

4 MGTP の並列化

MGTP の節は、前件 (リテラルの連言) と後件 (リテラルの選言) から構成されており、前件における連言照合操作の並列性を AND 並列、選言の場合分けによる並列性を OR 並列と呼ぶ。

MGTP の証明過程には、主に次の三つの並列性の源がある。

(1) 複数のモデル候補の探索

(2) 連言照合

(3) 包摂テスト

(1) は、後件部が複数のアトムから構成される非ホーン節でモデル拡張を行なった場合に発生する並列性である。これは複数のモデル候補 (証明木の各枝) を同時に探索することであり、与えられた節集合が値域限定であれば、他との交信なく独立に探索を行なえるので、OR 並列性とみな

process T:
  repeat forever
    $request(G, \Delta)$;
    $\Delta' := subsumption(\Delta, M \cup D)$;
    if $CJM_{tester}(\Delta', M \cup D) \ni false$ **then return**(unsat);
    $D := D \cup \{\Delta'\}$.

process G:
  repeat forever
    while $Buf = \phi$ **do begin**
      $D := D \setminus \{e\}$; $Buf :=$ **delay** $CJM_{generator}(e, M)$; $M := M \cup \{e\}$ **end**;
    $wait(T)$;
    $\Delta :=$**force** Buf;
  until $D = \phi$ **and** $Buf = \phi$.

図 3.5 遅延モデル生成アルゴリズム

すことができる[†2]。

(2) についてもう少し詳しく述べると、1) 各節の連言照合は独立に行なえ、さらに 2) 一つの節に対する連言照合も並列に実行できる。しかし、節中に共有変数がある場合、これらの変数は矛盾しない値をとらなければならない。後件部が一つのアトムのみからなるホーン節を扱う場合は、モデル候補が一つしか存在せず (証明木は一本の枝からなる)、連言照合が並列性の大きな源となる。この場合、ホーン節集合の全てのアトムは唯一つの解に関与するので、ホーン節に対する並列性は、AND 並列性とみなせる。

(3) の計算は、元来かなりの逐次性を含んでいる。モデル拡張候補 D 中のアトムに対する包摂テストを過不足なく完全に行なうためには、包摂テストの順序を固定する必要があり、あるアトムの包摂テストは、それ以前のアトムの包摂テストが完了しないかぎり、完了できない。しかし、次の包摂テストは前の包摂テストの完了を待たずに開始でき、ある所まで進めておけるので、並列実行はある程度可能である。しかし、依然この包摂テストは、いまのところ最も大きな並列実行阻害要因となっている。

非ホーン節を含む問題の場合には、複数の解を同時に探索する OR 並列性を利用することによって十分な並列効果が得られる。しかしながら、ホーン節のみからなる問題の場合はこのような OR 並列性はないので、AND 並列性を引き出さなければならない。AND 並列で台数効果を得るのは OR 並列に比べて格段に難しい。実際、MGTP の並列化の労力の大半は AND 並列化に投入された。以下に、OR 並列化方式と AND 並列化方式について述べるが、紙面の大半が AND 並列化に割かれているのは、このことを反映している。

5 OR 並列化方式

複数のモデル候補の探索を並列に行なう時には、どのプロセッシング・エレメント (PE) でどのモデル候補を探索するかを指示する方法 (PE 割付法) が、台数効果を高める上で重要である。この指示が適切でないと、PE の負荷が不均等になり期待していた効果が得られない。ここでは実際に我々が実験で用いた割付法を二、三紹介する。

最も安易な割付法は、ある PE で複数のモデル候補が得られたら、一つを残して残り全てのモデル拡張候補の探索を相異なる他の PE に割り当てていく、というものである。しかし、これには親元の環境のコピーが必要であり、また一般的にモデル候補数は爆発的に増えていくので、プロセス間通信が多発し現実的ではない。そこで、このように無闇矢鱈に PE を割り当てるのを抑制することが現実的な解法となる。我々はこの OR 並列探索の起動を抑制する方法を、有界 OR 並列と呼んでいる。

基本的な考え方は、利用可能 PE 数を埋めるのに充分な

†2 「後件部の変数は全て前件部に出現する」という値域限定性 (range-restrictedness) を節集合が満たす時は、生成されるアトムは変数を含まないので場合分けの際に共有変数の問題が生じず、証明木の各枝の探索は独立に行なうことができる。しかし、この性質を満たさない場合には共有変数の束縛値の一致性を検査する必要があり、単純な場合分けができず、AND 並列性と同種の問題が生じる。

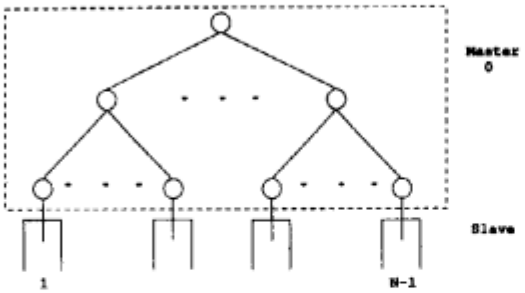

図 5.1 PE 割付法

プロセスを発生させた後は、他 PE に割り付けることなしに自 PE 内で以降の処理を行なうというものである。この方法は OR 並列探索では異なる枝間では通信が生じないという性質を利用したものであり、これによって PE 間通信が軽減できる。

一つの実現法としてはマスター・スレーブ方式が考えられる。マスター PE が空モデルから始めてモデル候補を拡張し、モデル候補の数が利用可能 PE 数を超えると、続きの探索をスレーブ PE に分配する、という方法である (図 5.1)。スレーブ PE は割り当てられたモデル候補を自分で探索するのみで、他の PE にさらにモデル候補を分配するようなことはしない。

また、マスターを利用しない実現法も考えられる。これはある深さに達するまでは、モデル候補の分岐の度に何らかの方法で (例えば modulo 計算によって) 負荷が均等になるように割り付け PE を決定し、その深さ以降は自 PE 内で処理を継続するというものである。

上記二つの方法は探索木が均衡している場合に適している。この場合、どのモデル候補の探索も大体同じ計算コストになるので、PE の負荷が均等になりやすい。またこれらの方法は、通信コストを最小化できるという点で簡便な方法である。

有界 OR 並列にこだわらない割り付け法としては、モデル候補の分岐の度に割り当てる PE を確率的に決定する方法もある。探索木が不均衡な場合は各枝の探索コストは予見できないので、このような確率的方法が有効になる。しかし、前にふれた通り、通信コストの点では問題が残る。

6 AND 並列化方式

一つのモデル候補の探索を並列に行なう AND 並列化では、OR 並列化に比べより細かな制御が必要になる。ここでは、一つの方式を提案するが、まずその方式を考える際に考慮した事項について述べる。そして、負荷分散法、粒度、並列性阻害要因とその対処法などについて触れた後、本方式の動作概要について述べる。最後に本方式を実現する際に用いた KL1 コーディング技術を簡単に紹介する。

6.1 設計指針

AND 並列化にあたって、以下の方式上のオルターナティブを検討した。

(1) PE 台数にかかわらず証明が変わらない証明不変方式と PE 台数に応じて変わる証明変動方式。
(2) モデル共有 (分散メモリアーキテクチャでは各 PE が同じモデル候補をコピーして持つ) 方式とモデル分散方式。
(3) マスターありとマスターなし。

証明不変方式は、モデル拡張候補 D からの Δ の取り出し順、アトムの生成順、包摂テストの順番を固定し、PE 数によらず同一の証明を得ようとするものである。一方、証明変動方式はこれらの順番を限定せず、先着順で処理する。従って、使用する PE 数が異なると得られる証明は変わる。

証明変動方式では、超線形台数効果が得られる可能性がある一方、使用した PE 台数に見合う台数効果が常に得られるとも限らない。一方証明不変方式では、使用した PE 台数に見合う台数効果が期待できるが、それは高々線形である。

モデル共有方式の利点は、連言照合や包摂テストといった最もコストのかかる計算を最小の PE 間通信で行なえることである。一方、一つのモデル候補を各 PE に分散配置するモデル分散方式では、メモリ・スケーラビリティを得ることができるという効果がある。しかしながら、生成されたアトムは包摂テストのために全 PE を一順しなければならないため、PE 台数が増えるにしたがって通信量が増大してしまう。

マスター・スレーブ方式では、逐次版 MGTP をスレーブ PE 上に置き、単にそれらとマスタ PE とをスター状につなぐことによって簡単に並列システムを構築することができる。しかし、マスターの負荷をできるだけ小さくする工夫が必要となる。逐次版 MGTP をリング状に結合するマスターなしの方式も考えられるが、各 PE を協調的に制御することは難しくなる。

並列定理証明器を開発する際の我々の方針は、得られた速度向上が並列化によるものなのか、戦略による探索空間の刈り込みによるものなのか、を明確に区別することである。証明変動方式においては、PE 台数を変えることは一

種の賭けを行うことに等しい。この場合、解がより速く得られるかどうかは実行させてみないと分からない。

以上のことを踏まえて、証明不変方式を基本とし、モデル共有、マスター・スレーブ型の AND 並列版 MGTP を遅延モデル生成法に基づいて実現した。

本システムでは、マスタープロセス及び複数の生成 (G) プロセス、包摂テスト (S) プロセス、棄却テスト (T) プロセスからなり、G プロセスと S プロセスは要求駆動的に動作し、T プロセスはデータ駆動的に動作する。動作概要については **6.5 AND 並列化方式概要** で述べる。このシステムの特徴は以下の通り。

(1) 証明不変方式なので、PE 台数に見合った速度向上が得られる。
(2) KL1 の同期機構により包摂テストの逐次性を最小にすることができる。
(3) スレーブプロセスは、暇になるとマスタープロセスからタスクを受けとれ、また各タスクの粒度はほぼ均等なので、良好な負荷分散が得られる。
(4) KL1 のストリーム通信により、要求駆動制御を容易にかつ効率良く実現することができる。

要求駆動制御により、不必要なモデル拡張や包摂テストを抑制でき、さらに高い稼働率を得ることができる。高稼働率の達成は線形台数効果得るためには必須である。

6.2 負荷分散

AND 並列の実現にあたっては、G、S、T の論理的プロセスを物理的 PE にいかに配置するかが重要な課題となる。この配置法は大まかにいって、(1) 一つの PE に一種類のプロセスしか配置しない「機能分散」法と (2) 各 PE に三種類のプロセスを重ねて配置する「負荷分散」法の二種類がある。

(1) の方法では、時々刻々と三種類のプロセスの負荷比率が変動する問題では、性能を上げるのは難しい。例えば、G プロセスが忙しい時間帯には S プロセスや T テストプロセスが配置されている PE は暇になる。このような場合には、G プロセスをより多くの PE に配置するために、既に配置されている他のプロセスをそれらの PE から排除するような操作が必要となるが、そのタイミングや再配置の度合を適切に決めることは非常に難しい。むしろこの方法が真価を発揮したのは、性能を上げるためのパフォーマンスデバッグの時であった。各 PE の稼働状況を視覚的に表示する rmonitor で観察することによって、時間と共に変化するプロセスの負荷比率が分かり、問題の傾向のみならず並列化方式の善し悪しも把握することができた。例えば、一つの PE だけ忙しくて他の PE は全く稼働していない極端な場面に遭遇することもあったが、これは方式あるいは実現手法に何らかの欠陥がある場合が多かった。このようなやり方で、性能劣化の要因となるボトルネックがいくつか明らかになり、性能改善を図ることができた。

さて、(2) の場合は三種類のプロセスを重ねて配置するわけであるから、(1) でのそれぞれの負荷が重ね合わせられることが期待される。G プロセスが暇な時は、T プロセスが穴を埋めてくれるようなことが期待されるわけである。これはプロセスの「動的変身」による動的負荷分散効果を狙ったものであるが、この方法ではプロセススケージューリングが性能上最も重要になってくる。緊急度の高いプロセス[11]がある PE で発生した時に、その PE では他のプロセスが走行しているためなかなか実行に移されず、その結果、他 PE のプロセスが idle 状態になる事態が発生する。これに対処するには、最も緊急度の高いプロセスから実行すれば良いわけであるが、この緊急度というのはシステム全体を見渡さないと分からないから制御は厄介なものになる。最適なスケージューリングは、机上だけではほとんど予想できないので、試行錯誤の連続で性能を上げていくことになる。このようなときに rmonitor のようなチューニングツールが有効であるのは言うまでもない。

6.3 粒度

G、S、T の各プロセスの処理の単位を、1 アトム (Δ) 毎とした場合の各プロセスの粒度を見積もってみよう。粒度が粗ければ並列性がでないが、細分化すると今度は分割によるオーバヘッドが生じるので、粒度の大小はシステムの性能を大きく左右する。

連言照合を行なう G、T プロセスの粒度は、節前件部のリテラル数に依存する。簡単のためリテラル数が 2 の場合を考えると、i 番目のアトムとそれ以前のアトム集合 $(1 \ldots i-1)$ との連言照合は $i \times (1 \ldots i-1) + (1 \ldots i-1) \times i + i \times i$ の掛け合わせを行なうので、その回数は $2(i-1)+1$ となり、粒度は i に比例したものになる。

i は G プロセスでは M の要素数に、T プロセスでは $M \cup D$ の要素数に相当するので、G プロセスの粒度は $|M|$ に比例し、T プロセスの粒度は $|M \cup D|$ に比例することを意味する。一般にリテラル数が n の時には G プロセスの粒度は $|M|^{n-1}$ に比例し、T プロセスの粒度は $|M \cup D|^{n-1}$ に比例する。

一方包摂テストは、すでに包摂テスト済みの要素 ($M \cup$

[11] そのプロセスの実行が終らないと他の (多くの) プロセスの実行を開始できないようなプロセスは緊急度が高いといえる。

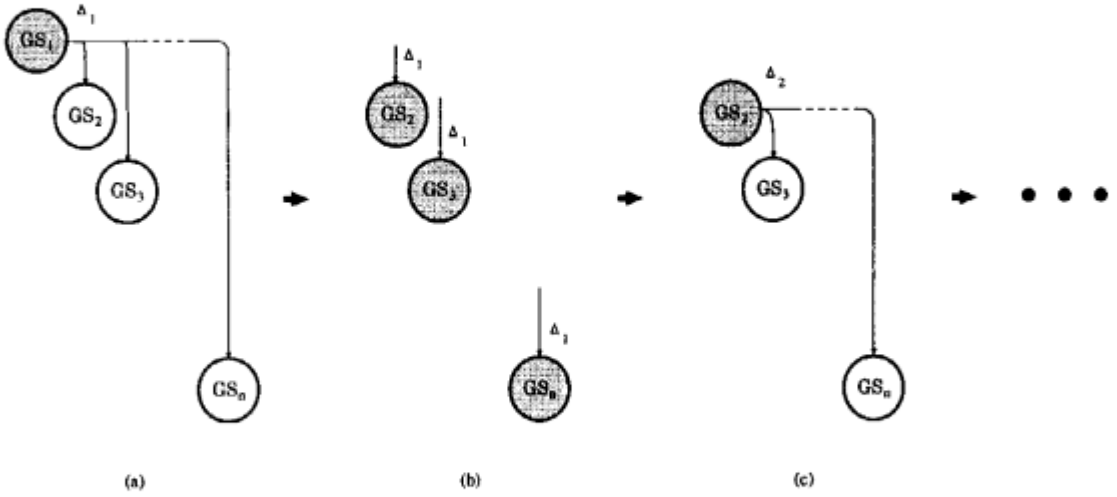

図 6.1 大域的包摂テストのパイプライン実行

D) に対して総当たりよる検査が必要になる。よってこの粒度は $|M \cup D|$ に比例する。

以上の考察は線形探査による操作を前提にしているので、何らかのインデックス機構を導入すれば 1 回当たりの仕事量は小さくなり得ることに注意されたい。

当初、1 アトム (Δ) を処理の基本単位とすれば十分な並列性が引き出せるものと考えていた。というのは、証明が進むにつれて $M \cup D$ が成長し、粒度もしだいに大きくなるからである。ところが、連言照合においてユニフィケーションに失敗すると、Δ に対するそれ以降の処理は行なわれなくなるので、このため負荷の不均一が生じ、$|M \cup D|$ が数千から数万になる問題では、それがより顕在化した。これとは逆に、計算量が大量な割には $M \cup D$ がなかなか成長しないような問題も幾つかあり、基本処理単位が 1 アトムでは、十分な並列性を引き出すことは困難であることが少しずつ明らかになった。この対処法については**次節並列性阻害要因** を参照されたい。

6.4 並列性阻害要因

本並列化方式には原理的に二つの逐次性があり、並列性能を追求する上での阻害要因となっている。一つは前にも少し触れたが包摂テストの逐次性であり、もう一つは証明不変を保証するための逐次性である。

A. 包摂テストの逐次性

包摂テストの逐次性は、直前のアトム (Δ''_k) の包摂テスト終了しないと、次のアトム (Δ''_{k+1}) の包摂テストが終了できないことに起因する (図 3.3 参照)。この逐次性は二項間の包摂関係に全順序性がないことから生じるものであり、互いに包摂しあわないアトム集合を得るには、勝ち抜き戦ではなくリーグ戦、つまり総あたりが必要となる。これは原理的な問題であり、実現手法によるものではない。

我々は、包摂テストを局所的包摂テスト (LS) と大域的包摂テスト (GS) に分離することにより、逐次性の低減を図った。生成されたアトムに対する包摂テストは、それ以前に生成されたアトム集合 A に対して行なわれなければならない。LS では、これを既に包摂テスト済みのアトム集合 $M \cup D$ に対しておこない、GS ではその残り $A \setminus (M \cup D)$ に対して行なう。

現方式ではモデル共有方式を採用しているので各アトムに対する LS は他に影響を与えることもなく、各 PE 内でそれぞれ独立に行なうことができる。GS には上で述べたような逐次性があるので、パイプライン実行により並列性を抽出している (図 6.1)。

いま、n 個の大域的包摂テストプロセス $GS_1, GS_2, \ldots, GS_n$ が動いているものとする。各 GS_i は受け持ちアトム Δ_i の処理中であるとする。GS_1 で行なわれていた Δ_1 の包摂テストが終ると、その結果が他の $GS_i (1 < i)$ に直ちに知らされる (a)。各 GS_i の動きは、この包摂テストの結果によって異なる。Δ_1 が包摂されなかったときは、Δ_1 に対して Δ_i の包摂テストを行ない、包摂された場合は特になにもしない (b)。いずれの場合も、次に GS_2 は Δ_2 の包摂テストの結果を他の $GS_i (2 < i)$ に知らせる (c)。このように、各 GS_i の待ちは、高々 (自分より以前の) プロ

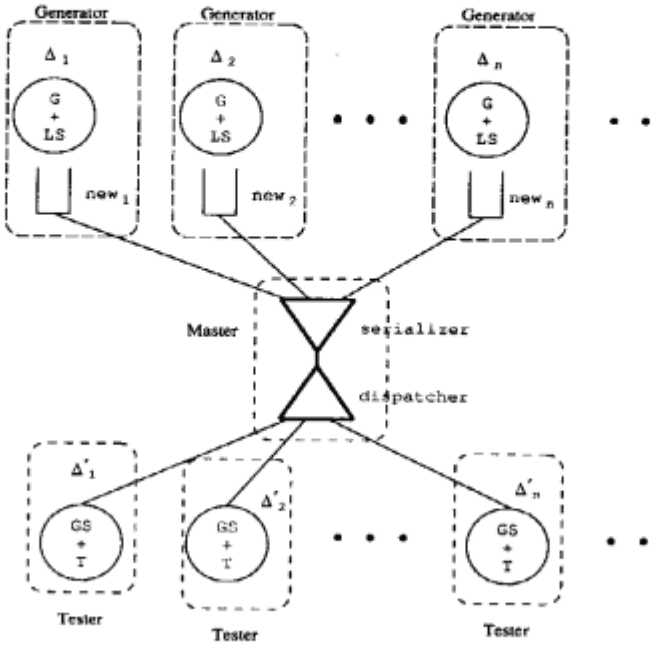

図 6.2 MGTP の AND 並列化

セス個数分だけで済む。

包摂テストの実装法としては、各項を木構造として「項メモリ」に保持し、共通部分項をくくり出すことによって検索効率向上をはかる方法と、各項をそのまま「リスト」形式で保持しリニアサーチによって検索する簡単な方法があるが、LS では項メモリによって、GS ではリニアサーチよって包摂テストを実現している。

B. 証明不変による逐次性

証明を不変にするためには、生成されたアトムの取り出し順序を固定しなければならない。この順序を固定することは、逐次性が発生することに他ならない。例えば、二つの G プロセス G1 と G2 が並列に動いている時、G1 の生成アトム集合 New_1 と G2 の生成アトム集合 New_2 は並列に生成されるが、早くできたアトムから取り出すわけにはいかない。どちらが早くできるかは非決定的であるから、予め決められた順序で取り出さなければ、実行のたびに証明が変わり得る。さて、予め New_1 から取り出すことになっていたとすると、New_1 からの取り出しが完了しなければ New_2 から取り出されることはない。したがって、New_1 の生成が何らかの理由で遅れた場合、New_2 がすでに生成されていたとしても、次のアトムを取り出すことができないので S や T プロセスが動けない状態になってしまう。もちろん、G1 の実行が滞っていたとしても、G2、G3 ... と次々に G プロセスを並列実行させれば見かけ上暇な PE をなくすことはできるが、これは無駄なアトムの生成を引き起こす可能性があり、遅延生成の考え方に反する。

実際に走行させてみても、ある PE の G プロセスが生成したアトムを取り出している間、他の PE の G プロセスは、指定個数のアトムを生成した後に、待ち状態に入り、各 PE の G プロセスの稼働状態が、先頭 PE から 最後の PE へと順次推移していく尺取り現象がみられた。

そこで上記の逐次性を低減するために、G プロセスの仕事の単位を k 分割し、連言照合 $CJM_{generator}(\Delta, M)$ を $|M|/k$ (k は定数) 個の G プロセスで行なうようにした。このようにすることにより、G プロセスの粒度がより細かくなり、かつ不均一さも顕著ではなくなり、PE の平均稼働率が 90 % 以上という性能向上に成功した。

6.5 AND 並列化方式概要

図 6.2 は、AND 並列化した際の各プロセスの結合関係を示したものである。中央のマスター (M) プロセスを介して上部の Generator プロセス群と下部の Tester プロセス群が連結している。

マスタープロセスの基本的な仕事は、Generator プロセ

スが生成したアトム (LS 済み) を Tester プロセス (GS も行なう) に分配することである。各スレーブプロセスは、暇になると次の仕事をもらうためにマスタープロセスに要求を出す。マスタープロセスは、1) 要求に先着順に応答し、2) 各スレーブプロセスが同じ仕事をしないように仕事を排他的に割り当て、3) Generator プロセスの生成したアトム数と Tester プロセスに分配したアトム数の差 (生成したが Tester プロセスによって消費されていないアトム数) を一定に保つことにより、Generator プロセスの過剰生成を抑制する。1) により自然に動的負化分散も行なわれる。

要求一回あたりのマスターの仕事量は、基本的には生成アトム Δ の配分しか行なわないので、非常に軽い。

各 Generator プロセスが生成したアトム列 new_i は、serializer によって順序付けられる。これは証明を PE 数によらず不変にするためである。この serializer を merger に取り替えるだけで証明変動方式の MGTP を容易に作ることができる。serializer で整列したアトムは、dispatcher によって Tester プロセスに分配される。

Generator、Tester、Master プロセスの概略手続きは次の通りである。

Generator: Master に要求を出し、generation 番号 gnum を受けとる。この generation 番号より、割り当てられた Δ の番号 i、k 分割のブロック番号 B_j、及び generator 節の内今回生成に用いられる generator 節集合 GCL を計算する。これを元にモデル拡張 $CJM^{B_j}_{GCL}(i,1\ldots i-1)$ を行なう。生成された各アトムについて局所的包摂テスト LS を行ない、包摂されなかったアトム集合 New とその要素数 num を Master の serializer に送る。なおここで、$CJM^{B_j}_{GCL}(i,1\ldots i-1)$ は $CJM_{GCL}(i,1\ldots i-1)$ を k 分割した部分集合である。

Tester: Master に要求を出し、Δ' 及び整数 l を受けとる。Δ' は大域的包摂テスト GS で包摂されない場合はモデル拡張候補 D に登録されるが、l はその時の登録番号を示している。まず、Δ' に対して GS を行なう。包摂された場合は l を M に返す。包摂さなかった場合は l 番目の要素として Δ' を登録して、$l+1$ を M に返し、棄却テスト $CJM_{tester}(l,1\ldots l-1)$ を行なう。$CJM_{tester}(l,1\ldots l-1) \ni false$ なら、全プロセスに中断指令を送る。

Master: 初期値として $l := gnum := 1, enum := 0, queue := \phi, limit :=$ ある正整数 とする。

1) Generator から要求があった場合: $enum < limit$ になるまで待つ。$gnum$ を Generator に送り、$gnum := gnum + 1$ とし、Generator が計算した New を $queue$ の末尾に連結する。また、$enum := enum + num$ とする。
2) Tester から要求があった場合: $enum := enum - 1$ とし、$queue$ の先頭から要素 Δ' を一つ取りだす。そして、残りを $queue$ とする。Δ' と l を Tester に送り、Tester から返ってきた l' を l とする。

ここで、$enum$ は、Generator で生成されたが、Tester に送られていない要素数の大まかな値を示している。Master プロセスは、Generator プロセスから要求があったとき、$enum < limit$ になるまで要求を後回しにすることにより、Generator プロセスの先行し過ぎによるアトムの過剰生成を防いでいる。

包摂テストを専門に行なう S プロセスが、Generator プロセスにおける LS と Tester プロセスにおける GS に分離されている点に注意されたい。S プロセスを LS と GS に分けることは、S プロセスの逐次性を低減する他に、LS において生成要素のふるい落としが行なわれることによる PE 間通信の低減も計れる利点を持つ。

6.6 KL1 コーディング技術

MGTP の並列化にあたっていくつかの KL1 コーディング技術が比較検討された。

A. インプリシット・コピーイング

情報を PE にまたがって共有したい場合 (モデルコピー方式)、(1) その情報の複製／転送をプログラムの上で、ストリームを用いて陽に記述するか、もしくは、(2) 構造データの頭 (D-リストの先頭) を各プロセスに共有させ、必要な項目まで辿るのはファームレベルの動作に任せるか、のいずれかがある。

(1) の方式が優れていると考えられるのは、転送すべき情報のひとまとまりが本当に一度に PE 間を渡ることが保証されているときである。特に、ある PE に所在している構造データを他の複数の PE に転送したいとき、ストリームでチェイン接続した PE を順番に転送するように書けば、発信元の PE に対するメモリ競合の問題がなくなる。さらに、発信元が複数あれば、異なるタイミングで複数の情報が異なる PE 間で同時に転送でき、ネットワークのスループットが向上する。一方、(2) の方式によると、発信元の PE に対するメモリ競合の問題があり、台数の増大とともにボトルネックが生じやすくなる。

ところが、KL1 では 2 以上の深さをもつ構造データ (ベクタ) が一度に転送されることはないので、(1) の方式でも

必ずしもメモリ競合の問題がなくなるとはいえない。構造データを平坦な別のデータ形式 (例えばストリング) に変換して転送すればよいが、変換のコストが無視できない。

(2) の方式は、元来共有メモリ型のアーキテクチャに適したものであるが、分散メモリ型の PIM のアーキテクチャにおいても (1) に比べて極めて悪いということはない。むしろ、通信記述にともなうオーバヘッドが軽減されると考えられる。また、(2) の方式では、PIM のようなローカルメモリ型のマルチプロセッサ上であたかも共有メモリを使用しているようなコーディングが可能となる。これは、KL1 プログラミングの大きなメリットであり、これにより、コーディング作業量及び開発期間が大幅に短縮された。

B. 優先度制御

1 つの PE に異種のプロセス群を配置する負荷分散方式では、プロセスのスケジューリングが重要であると述べた。しかし、KL1 のシステムスケジューラを直接操作することはできないので、ゴールの優先度を制御することによりスケジューリングを間接的に行なっている。

さて、図 6.2 に示した AND 並列化方式では、実際には Generator は、モデル拡張 (G) と局所的包摂テスト (LS) をまとめて行なう一つのプロセスとして、一方 Tester は、大域的包摂テストを行なう GS プロセスと棄却テストを行なう T プロセスの二つのプロセスで構成されている。これらのプロセスにどのような優先度を与えるかについて考えてみよう。Generator プロセスがアトムを生成しないと他のどのプロセスも動けないという点で Generator プロセスの緊急度は高いが、不要なアトムを作り過ぎると遅延生成の原則に反することになる。GS プロセスは、ある要素の包摂テストが終了しないと他の PE で行なっている包摂テストが終了しないし、棄却テストも開始できないのでこれも他のプロセスへの影響が大きい。T プロセスについては、それが終了しないからといって他のプロセスの実行が開始できないというわけではないので緊急度は低いが、いわゆる全計算の終了判定を行なっているので実行をいつまでも後回しにするわけにもいかない。

また、実際には複数の Generator プロセスと複数の GS 及び T プロセスを一つの PE に割り付けているので、同種の複数のプロセス間の優先度も考えなければならない。これには以下に述べるスライド優先度割当法を用いている。

すなわち、同種のプロセス間では、個々のプロセスは降順の優先度を付加されて起動される。ある優先度 Pri をもつプロセスの計算が終了したら、Pri 以下の優先度が付加されたプロセスの優先度を一つずつ上げる。これにより、最低優先度 Pri_{min} をもつプロセスの優先度も一つ上がり、次に起動するプロセスの優先度として Pri_{min} を割り当てることができる。このように優先度を再利用することによって、有限の優先度しか提供されていない KL1 処理系で、あたかも無限の優先度があるかのような環境を模擬することができる。

以上のことを踏まえ、優先度は高い順に 1) GS プロセス、2) Generator プロセス、3) T プロセス、としている。ただし、このような優先度では GS プロセスと Generator プロセスのみで暴走する可能性がある。そこで、1 PE あたりの未終了の T プロセス数がある閾値を越えると、これらの T プロセスの幾つかの実行が終了して閾値の範囲に収まるまで GS プロセスの起動を遅らせるようにしている。このようにして、GS、Generator プロセスによる暴走を防いでいる。

C. 要求駆動制御

KL1 には元来、必要なデータが揃うと実行を開始する同期機構が備わっており、要求駆動を実現するには適したプログラミング言語である。

遅延モデル生成の心は、新しいアトムが必要な時にのみ生成を行なう、つまり、G プロセスを動かせばよいというものだから、これは G プロセスを要求駆動に動作させることにより実現できる。

G プロセスにおいて、Δ とそれ以前のアトム集合 M との連言照合を行なう最内側ルーチンは、要求駆動制御になっていなければ、大体次のような KL1 プログラムになる。

本ルーチンは連言照合に必要な Δ と M の要素の重複を許した組合せパターンと節パターンの組 (Cmb) を受けとる。

```
g([], Out) :- % 全ての連言照合が終了したら、
  Out = [].   % 出力ストリームを閉じる。
g([Cmb|Cmbs], Out) :-
              % あるパターン Cmb について
  tryCJM(Cmb, Rst),  % 連言照合を試みる。
  (Rst = success ->  % 成功したら
     newConc(Cmb, Conc),
              % 後件部 Conc を生成して
     Out = [Conc|Out1],
              % 出力ストリームに流す。
     g(Cmbs, Out1);
   Rst = fail -> g(Cmbs Out)).
              % 失敗したら何もしない。
```

これに、要求を伝達するストリーム Dmds を加えて、

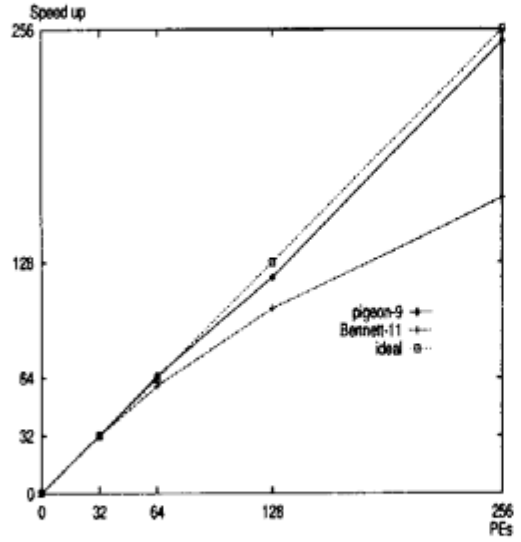

図 7.1 速度向上比 (OR 並列)

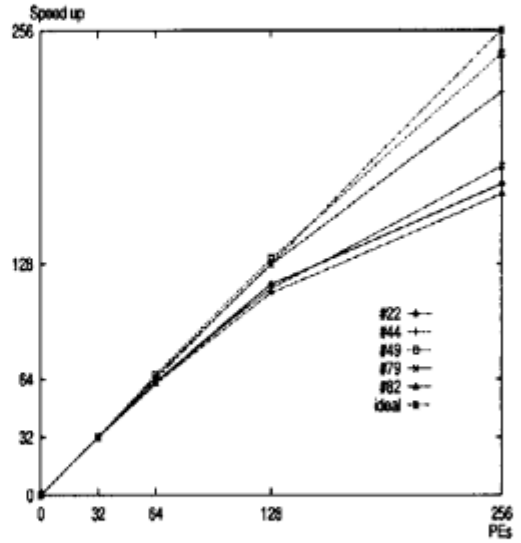

図 7.2 速度向上比 (AND 並列)

```
g([Dmd|Dmds],[Cmb|Cmbs], Out) :-
                        % 要求があったら
  tryCJM(Cmb, Rst), % 連言照合を試みる。
...
```

という風にすると要求駆動制御になる。

実際には、このプログラムのように本当に必要になってから作り始めたのでは遅いので、ある程度のバッファリングが必要である。バッファが空にならないように、かといって作り過ぎないような微妙な制御が必要になる。この制御もマスタープロセスの大きな役割である。

7 実行結果

以上述べてきた MGTP の AND/OR 並列化方式の効果を PIM/m 256 台を用いて計測した。

7.1 OR 並列化

OR 並列実行による速度向上効果を見るため、pigeon hole と Bennett 問題 [7] の性能測定を行なった。pigeon hole 問題は、定理証明の分野においてベンチマークとしてよく用いられている。Bennett 問題は有限準群に関するもので、ラテン方陣の数え上げ問題と見なすことができる。いずれも生成されるアトムは基底アトムのみであるが、OR 分岐の数が組合せ的に増大する問題である。

証明図の形状に関していえば、pigeon hole 問題は良く均衡しており、Bennett 問題は均衡の度合が悪い。したがって Bennett 問題では、探索する証明の枝が不均一なので、pigeon hole 問題に比べると負荷を均等に割り当てるのが困難である。図 7.1 は、速度向上比をグラフ化したものである。pigeon hole 問題 (hole 数 9) ではほぼ理想線に近く、また負荷の均等割当が困難な Bennett 問題 (ラテン方陣の位数 11) でも 170 倍程度の良好な速度向上が得られている。

7.2 AND 並列化

表 7.1 は、[15] に述べられている分離の規則に関する問題 (condensed detachment problem) 112 題の内、5 題に対する PIM/m の性能測定の結果を示している。図 7.2 は、これらの問題に対する速度向上比をグラフ化したものである。

分離の規則に関する問題というのは、いくつかの単位正節 $true \rightarrow p(A)$ と分離の規則を表す混合節 $p(X), p(i(X,Y)) \rightarrow p(Y)$ と一つの単位負節 $p(G) \rightarrow false$ からなる。混合節の後件部は連言を含まないので、枝分かれによる OR 並列性は全くない。

これらの問題の証明は、OTTER [16] で様々な戦略を用いて試されている。これらの戦略には 1) 項の重さによるソーティング、2) 1) と幅優先探査を混合して用いる、3) ある種のパターンを含むアトムを捨てる、等がある。

表 7.1 PIM/m 上での性能測定

問題	$\mid M \cup D \mid$	32 PEs(Kred/sec)	64 PEs	128 PEs	256 PEs
22	36497	23713557/8647.60	23722337/4437.82	24055288/2374.61	24847903/1613.99
44	15071	33365849/9688.86	33130370/4750.05	33601534/2429.84	33495763/1395.62
49	20622	63362335/18624.87	63031735/8948.01	63903440/4563.52	63036138/2450.42
79	14249	7982121/2523.79	8007157/1291.49	8160272/707.22	8425595/445.65
82	15103	5733605/1926.14	5767048/1004.32	5891856/551.93	6237114/371.37

MGTP では、幅優先探査を基本として、項の重さによる足切りと 3) を用いて証明を行なった。

MGTP の単一 PE での性能は、遅延モデル生成アルゴリズムによる計算量の低減もあり、Sparc-2 上の OTTER に比べ 2 ～ 5 倍程度高速である。

問題 44、49 の証明にかかる時間は、32 PE でそれぞれ 9700 秒、18600 秒であり、問題 79、82 の 2500 秒、1900 秒に比べかなり時間を要している。問題 22 の証明時間は 8600 秒と問題 44 と大差ないが、$\mid M \cup D \mid$ の大きさは、問題 44 では 15100 に対し問題 22 では 36500 と倍以上である。

台数効果をみると、問題 44、49 では約 230 倍以上、一方問題 82、22、79 では 170 から 180 倍程度の速度向上が得られている。

問題 44、49 の台数効果が他の問題に比べて優れていることから、時間を要する問題ほど、また $\mid M \cup D \mid$ の成長速度 ($\mid M \cup D \mid$ /sec) が遅い問題ほど良好な台数効果が得られていることが分かる。$\mid M \cup D \mid$ の成長速度が速いと、1) 大域的包摂テストにかけられるべきアトムと 2) 各 PE で貯えられるアトムの単位時間あたりの数が多くなる。これによって、1) 大域的包摂テストの逐次性が顕著になり、2) 単位時間あたりのアトム転送頻度が多くなる。この逐次性と転送ネックが、$\mid M \cup D \mid$ の成長速度が速いと並列性能が劣化する原因として考えられる。

全体として、128 PE 以上では、グラフの傾きがやや緩やかになる傾向が見られるものの、256 PE までては速度向上の飽和現象は見られない。ほぼ、線形な台数効果が得られ良好な結果を示しているといえる。

8 むすび

第五世代コンピュータ・プロジェクトで開発した並列定理証明系 MGTP について、並列化方式とその実現手法を中心に紹介した。本研究では、大規模並列定理証明系の優位性を示すため、PIM 上で「線形に近い台数効果」を達成し、これまで逐次マシンでは計算時間・メモリ容量の観点から不可能であった「ハードな問題」に挑戦し、これを解くことを第一目標に掲げた。主に、分離の規則に関する問題や準群問題のような数学的定理証明の問題に取り組んできたが、いわゆる未解決問題を解くことにも成功するなど、当初予想してなかったような成果が得られた。しかし、並列化方式の観点からは、解決すべき課題もいくつか浮き彫りにされてきている。

現在の AND 並列化方式は 256 PE 上で 200 倍以上の速度向上を達成し、かなり良好な並列性能を示しているが、包摂テストの逐次性の問題は完全には解決しておらず、スケーラビリティの点で限界がある。今後、項の重さによるソーティング等の機能を組み込む予定であるが、同種の問題が予想される。また、ホーン節、ノンホーン節対応にそれぞれ個別に AND 並列化方式及び OR 並列化方式を開発したが、一般にはこれらの節が混在する場合が多く、AND 並列性と OR 並列性の両方を同時に取り出すことができる AND/OR 統合方式を確立する必要がある。

MGTP 研究の最終目標は、論理プログラミング技術と自動推論技術を融合することである。MGTP は KL1 が失った全解探索機能を自然な形で回復しており、従来のトップダウン型とは異なる、ボトムアップ型をベースとする新しい論理プログラミングパラダイムとなる可能性を持っている。今後、AI 応用向けに MGTP の機能拡張を進めると共に、MGTP の特長を生かした応用領域を開拓していく予定である。

参考文献

[1] M.E. Stickel. A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler. *Journal of Automated Reasoning*, 4:353–380, 1988.

[2] R. Manthey and F. Bry. SATCHMO: a theo-

rem prover implemented in Prolog. In *Proceedings of the 8th Conference on Automated Deduction*. Springer Verlag, 1987.

[3] 潮 一博. KL1 プログラミング雑感 - prover の並列化の体験より -. In *Proceedings of KL1 Programming Workshop' 90*, pages 131–139. ICOT, 1990.

[4] R. Hasegawa, H. Fujita, and M. Fujita. A Parallel Theorem Prover in KL1 and Its Application to Program Synthesis. In *Proceedings of Italy-Japan-Sweden Workshop '90*. ICOT, 1990. ICOT TR-588.

[5] H. Fujita and R. Hasegawa. A Model Generation Theorem Prover in KL1 Using Ramified-Stack Algorithm. In *Proceedings of the 8th International Conference on Logic Programming*, pages 535–548, 1991. ICOT TR-606 1990.

[6] 藤田 正幸、J. Slaney、長谷川 隆三. 並列推論マシン上の並列定理証明系による有限代数の新事実 - 概要 -. TM 1221,1222, ICOT, 1992.

[7] M. Fujita, J. Slaney, and F. Bennett. Automatic Generation of Some Results in Finite Algebra. Proceedings of IJCAI-93 に掲載予定, 1993.

[8] 藤田 正幸、粂野 文洋. Mgtp による有限代数の新事実の発見. In *JSPP'93* 論文集, 1993.

[9] 長谷川 隆三、越村 三幸. モデル生成型証明器 MGTP の and 並列化方式. ICOT TR 申請中, 1993.

[10] R. Hasegawa and M. Fujita. Parallel Theorem Provers and Their Applications. In *Proceedings of the International Conference on Fifth Generation Computer System*, pages 132–154. ICOT, 1992.

[11] R. Hasegawa, M. Koshimura, and H. Fujita. Lazy Model Generation for Improving the Efficiency of Forward Reasoning Theorem Provers. In *Proceedings of the International Workshop on Automated Reasoning*, pages 191–202, 1992. ICOT TR-751.

[12] K. Inoue, M. Koshimura, and R. Hasegawa. Embedding Negation as Failure into a Model Generation Theorem Prover. In *Proceedings of the 11th International Conference on Automated Deduction*, pages 400–415. Springer Verlag, 1992. LNAI 607.

[13] 越村 三幸、長谷川 隆三. モデル生成型証明器上の様相命題タブロ. In *Proceedings of the Logic Programming Conference '91*, pages 43 52. ICOT, 1991.

[14] K. Inoue, Y. Ohta, H. Hasegawa, and M. Nakashima. Bottom-Up Abduction by Model Generation. TR 816, ICOT, 1992. Proceedings of IJCAI-93 に掲載予定.

[15] W. McCune and L. Wos. Experiments in Automated Deduction with Condensed Detachment. In *Proceedings of the 11th International Conference on Automated Deduction*, pages 209–223. Springer Verlag, 1992. LNAI 607.

[16] W.W. McCune. *OTTER 2.0 Users Guide*. Argonne National Laboratory, 1990.