TR-905

モデル生成型定理証明系の AND並列化方式

越村　三幸，長谷川　隆三

January, 1995

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系の AND 並列化方式

越村 三幸　長谷川 隆三
(財) 新世代コンピュータ技術開発機構 研究所

概要

一階述語論理の定理証明系 MGTP(Model Generation Theorem Prover) の AND 並列化方式を提案しその評価を行なう．MGTP はモデル生成法を証明手法として採用しており，並列記号処理言語 KL1 で記述されている．モデル生成法における定理証明過程では，一般に場合分けによる分岐を含み，分岐による並列性 (OR 並列性) を内在しているが，本論文では，このような分岐が全くない場合の並列性 (AND 並列性) 抽出を扱う．本方式では，モデル生成法を generate-and-test 型の計算としてモデル化し，そこに要求駆動の制御を導入して，並列環境下での負荷の均衡を計っている．方式評価は，分離の規則に関する問題を例に，256 台のプロセッサからなる分散メモリ型並列マシン PIM/m 上で行ない，200 倍以上の台数効果を得た．

1　まえがき

自動定理証明は，一種の記号処理計算で膨大な計算時間とメモリ空間を必要とする．このような計算を行なうためには，並列化による高速化が重要で，これまでにも幾つかの研究 [8, 10] がなされている．しかし，従来の研究は高々数十台規模の並列マシン上での実験に留まっており，数百台規模の並列マシンでの評価は皆無であった．また，多くの実験は共有メモリ型並列マシン上で行なわれており，大域的データへの参照及び変更が頻繁に必要になる定理証明系の分散メモリ型並列マシン上での実現は，通信やコピーがボトルネックとなり，困難とされていた．

本論文では，モデル生成型定理証明系 MGTP[2] の AND 並列化方式を提案する．本方式では，生成途中のモデル候補を各 PE にコピーし保持することにより，証明の基本手続きである連言照合や包摂テストを各 PE で局所的に行なうことを可能とし，通信とコピーの量の低減を図っている．

証明手法としてモデル生成法を採用した理由は，以下の通りである．1) モデル生成法は，ボトムアップ型の定理証明法の一種で，補題化，包摂テスト，削除戦略の組み込みが容易に行なえる．2) ボトムアップ型定理証明は，大量の大域的メモリへのアクセスを前提としたヒープ (メモリ) 指向の計算モデルに基づいており，同様にヒープ指向の計算モデルに基づいた KL1 言語向きアーキテクチャを持つ PIM/m 上に実装するのに適している．これに対し，モデル消去法に代表されるトップダウン型定理証明は，スタック指向の計算モデルに基づいており，スタックの伸縮が高速に行なえるマシン上での実装に適していると考えられる．

MGTP は，節集合で与えられた問題の充足可能性をモデル生成法を用いて計算する．ノンホーン節を含む節集合が値域限定性 [4] を満たす場合には，ノンホーン節の場合分けから生じる証明木の各分枝を独立に探査することによって，並列性 (OR 並列性と呼ぶ) が比較的容易に引き出せ，分散メモリ型並列マシン上で良好な結果を得ることができる [2, 12]．本論文では，このような分岐が生じないホーン節のみからなる節集合が与えられた場合の並列性 (AND 並列

性と呼ぶ) 抽出について考える．この場合，MGTP の証明手続きは，単位節のみを導出する超導出法 [1] と同じになる．

AND 並列性を抽出するために，我々は MGTP の証明過程を生成プロセスが次々に新しいモデル候補要素 (導出節) を作りだし，テストプロセスによってそのモデル候補が棄却されるか否かをテストする，generate-and-test 型の計算としてモデル化した．

また並列環境下での，生成プロセスの暴走による要素の過剰生成を抑えるために，モデル生成法に要求駆動の概念を導入した遅延モデル生成法 [3] をベースとした．テストプロセスはデータ駆動，生成プロセスは要求駆動で動作する．これにより，生成プロセスの暴走を抑制することができ，さらに生成プロセスとテストプロセスの処理速度を均一に保った並列・パイプライン処理が可能となった．

以下では，先ずモデル生成法について説明し，その逐次アルゴリズムとナイーブな並列アルゴリズムを示す．続いて AND 並列化にあたっての設計指針を述べ，要求駆動制御を採り入れた改良版並列アルゴリズムを示す．そして，分離の規則に関する問題を例に PIM/m-256PE 上で本並列アルゴリズムの定量的評価を行なう．

2 モデル生成法

MGTP はモデル生成法に基づいた一階述語論理の並列定理証明系で，並列記号処理言語 KL1[11] で記述されている．本節では，モデル生成法の概略について述べる．

MGTP の入力節は次のような節形式の集合として与えらる．

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

ここで，$A_i (1 \leq i \leq n)$ および $C_j (1 \leq j \leq m)$ はアトムである．$\rightarrow$ の左側を前件部，右側を後件部という．前件部は $A_1, A_2, \ldots, A_n$ の連言 (','), 後件部は $C_1, C_2, \ldots, C_m$ の選言 (';') である．$n = 0$ のとき，前件部を特に $true$ と書き，正節と呼ぶ．一方 $m = 0$ のとき，後件部を特に $false$ と書き，負節と呼ぶ．それ以外の節 $(m \neq 0, n \neq 0)$ は混合節と呼ぶ．また，$m \leq 1$ の節をホーン節，$m > 1$ の節をノンホーン節と呼ぶことにする．

モデル生成法は，与えられた節集合 S に対するモデル[1]を，空集合から始めて構成的に求める証明手法である．

モデル生成法には次の二つの規則がある．規則中 $\mathcal{M}$ は構成途中のモデル (これをモデル候補と呼ぶ) の集合を表す．$\mathcal{M}$ の初期値は $\{\emptyset\}$ である．

- モデル拡張規則: あるモデル候補 $M \in \mathcal{M}$ とある置換 σ があり，S 中の混合節もしくは正節 $A_1, A_2, \ldots, A_n \rightarrow C_1; C_2; \ldots; C_m$ の各前件アトム $A_i\sigma$ が M で充足されており，いずれの後件アトム $C_j\sigma$ も M で充足されていないとき，各 $C_j\sigma$ を M に加え (モデル候補の拡張)，M の代わりに $\mathcal{M}$ の要素とする $(\mathcal{M} := \mathcal{M} \cup \bigcup_{j=1}^{m} \{M \cup \{C_j\sigma\}\} \setminus \{M\})$.
- モデル棄却規則: あるモデル候補 $M \in \mathcal{M}$ とある置換 σ があり，S 中の負節 $A_1, A_2, \ldots, A_n \rightarrow false$ の各前件アトム $A_i\sigma$ が M で充足されているとき，M を棄却する $(\mathcal{M} := \mathcal{M} \setminus \{M\})$.

二つの規則のいずれも適用できなくなった時点で，S のモデルが $\mathcal{M}$ の要素として得られる．モデル生成法は健全で完全であるから $\mathcal{M} \neq \emptyset$ なら S は充足可能であり，$\mathcal{M} = \emptyset$ のとき，その S が充足不可能であることがわかる．

[1] ここでモデルとは，節集合 S を充足するアトムの集合のことである．

```
(0)  Init: MD[i] := C_i for i = 1,...,k where {C_1,C_2,...,C_k} = {C | (true → C) ∈ S}
(1)  Init: g := 1; s := k;
(2)  while g ≤ s do begin
(3)      foreach A_1,...,A_n → C
(4)          foreach (i_1,...,i_n) s.t. ∀j(1 ≤ i_j ≤ g) and ∃j(i_j = g)
(5)              if ∃σ(置換)∀j(MD[i_j]σ = A_jσ and Cσ is new) then begin
(6)                  s := s + 1; MD[s] := Cσ; end; /* モデル拡張 */
(7)                  foreach A_1,...,A_m → false do
(8)                      foreach (i_1,...,i_m) s.t. ∀j(1 ≤ i_j ≤ s) and ∃j(i_j = s) do
(9)                          if ∃σ'(置換)∀j(MD[i_j]σ' = A_jσ') then return (unsat);
                                 /* モデル棄却 */
(10)             end;
(11)     g := g + 1;
(12) end ;
(13) return (sat);
```

図 1: MGTP の逐次アルゴリズム

3 MGTP のアルゴリズム

ノンホーン節によるモデル拡張では，枝分かれ[2]が生じ，それぞれの枝の探索を並列に行なうことができる．枝分かれによる並列性を OR 並列性と呼ぶことにする．MGTP の OR 並列化の効果については，[2] と [12] に詳しい．

一方，本論文では専ら一つの枝の探索の並列性について考察していく．この並列性を AND 並列性と呼ぶことにする．本節で示すアルゴリズム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．

3.1 逐次アルゴリズム

MGTP で採用しているモデル生成アルゴリズムを図 1に示す．本アルゴリズムは，与えられた節集合 S が充足可能なら sat を，充足不能なら unsat を返す．

ここで，MD はモデル拡張の適用によって生成された (後件) アトムを保持するための配列であり，$MD[1]$ から $MD[g-1]$ までがモデル候補を，$MD[g]$ から $MD[s]$ までがモデル拡張候補 (モデル候補に付加されるべき生成アトムの集合) を表す．モデル候補は空集合，モデル拡張候補は正節の後件アトムの集合で初期化される ((0)，(1))．

(3) から (6) までがモデル拡張に対応している．各混合節の前件部がモデル候補で充足されているか否かを判定し ((4) と (5) の条件の前半)，充足可能であれば後件部の包摂テストを行ない ((5) の条件の後半)，包摂されなければ MD に加える ((6))．

(5) の条件部中の $C\sigma$ is new は，生成アトム $C\sigma$ が MD のどの要素にも包摂されない ($\nexists\sigma'$(置換)$C\sigma = MD[i]\sigma'$ for $\forall i(1 \leq i \leq s)$) ことを表す．包摂テストについては，本論文ではこの前方包摂テストのみを扱い，$C\sigma$ が MD の要素を包摂している ($\exists i(1 \leq i \leq s)\ |\sigma'$(置換)$C\sigma\sigma' = MD[i]$) かを調べる後方包摂テストは扱わない[3]．

その後の (7) から (9) がモデル棄却に対応する．各負節の前件部がモデル候補またはモデル拡張候補で充足されているかを判定し，充足されていれば証明手続きは終了する．

---

[2] 一つのモデル候補から複数のモデル候補が生まれること

[3] $C\sigma$ に包摂される $MD[i]$ を消去しなければならないので，データの一貫性を保つ必要が生じ，前方包摂テストより並列化は難しい．また，破壊代入を許していない KL1 では，消去の際にコピーが必要になる．

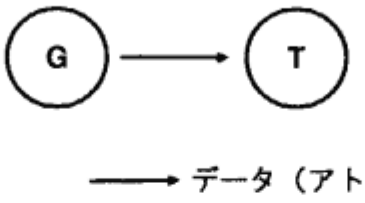

図 2: 生成 (G) と棄却テスト (T) プロセス

ここで，モデル拡張の際の混合節前件部の充足可能性の判定にはモデル候補が使われているのに対し，モデル棄却の際の負節前件部の充足可能性の判定にはモデル候補とモデル拡張候補の双方が使われていることに注意されたい．負節の前件部判定の範囲を混合節よりも広げることは，時間及び空間計算量の多大の削減をもたらす [3].

このアルゴリズムで基本となる操作は，節の前件部が MD 中のアトムで充足可能であるか否かの判定 (これを以降，連言照合操作と呼ぶ) と生成されたアトムの包摂テストである．MGTP の AND 並列化は，これら二つの操作の並列化に他ならない．

3.2 並列アルゴリズム

図 1の逐次アルゴリズムの並列化で最も簡単な方法は **while** ループの一つずつを並列に実行することである．ループ内部では，先ず混合節の連言照合 ((4) と (5) の前半)，包摂テスト ((5) の後半)，MD の更新 (6)，そして負節の連言照合 ((7) から (9)) が行なわれる．

我々は，これを混合節の連言照合を行なう生成プロセス (G プロセス) と負節の連言照合を行なう棄却テストプロセス (T プロセス) の二種のプロセスが図 2のように連結している generate-and-test 型計算と捉えてループ内部の並列アルゴリズムを考案した．

ここで，問題となる点は，並列に行なわれる包摂テストと MD の更新である．包摂テストおよび MD の更新は競合的に行なわれるので，ある種のロック機構が必要となる．

そこで，G プロセスが新たに生成したアトムを一旦蓄えておくバッファを設けることにした．図 3にそのバッファ (Buf で表現) を用いた並列アルゴリズムを示す．Buf を介して G プロセス群が生成したアトムは T プロセス群に送られる (図 4)．G プロセスは (G0) から (G6) までを，T プロセスは (T0) から (T8) までを繰り返す．G プロセスによって Buf に生成アトムが入れられ ((G5))，T プロセスによって Buf からそれらが取り出される ((T0))．

複数の G プロセスの排他制御は g に対するロックにより行なわれる ((G0))．(G1) から (G3) までの手続きは，逐次と変わらない．同様に (T4) から (T6) の処理も逐次と全く同じである．

包摂テストは，T プロセスで行なわれる ((T2))．MD は時々刻々更新されていくので，包摂テストを完全に行なうためには，MD をロックする必要がある．MD をロックした ((T1)) 後，Buf から取り出したアトム δ の包摂テストを行ない ((T2))，包摂されなければこれを MD に登録し，ロックを解除する ((T3))．

この並列アルゴリズムは，ロック操作が多用されており，この部分をどのように実現するかが性能上問題となるが，これについては 5節で述べる．また，一般に generate-and-test の計算では，generator の暴走によるアトムの過剰生成の危険性がある．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 [3] によって実際の実装を行なっている．

```
(0)(1)  図 1(0)(1) と同じ
(2)  Init: Buf := ∅; b := 1;

process G
(G0) lock g; g' := g; g := g + 1; unlock g;
(G1) foreach A_1, ..., A_n → C do
(G2)     foreach (i_1, ..., i_n) s.t. ∀j(1 ≤ i_j ≤ g') ∧ ∃j(i_j = g') do
(G3)         if ∃σ∀j(MD[i_j]σ = A_jσ) then begin
(G4)             lock Buf;
(G5)             Buf[b] := Cσ; b := b + 1;
(G6)             unlock Buf; end;

process T
(T0) lock Buf; δ := Buf[b]; b := b − 1; unlock Buf;
(T1) lock MD;
(T2) if δ is new then begin
(T3)     s := s + 1; MD[s] := δ; s' := s; unlock MD;
(T4)     foreach A_1, ..., A_m → false do
(T5)         foreach (i_1, ..., i_m) s.t. ∀j(1 ≤ i_j ≤ s') ∧ ∃j(i_j = s') do
(T6)             if ∃σ'∀j(MD[i_j]σ' = A_jσ') then
(T7)                 return (unsat);
(T8) end else unlock MD
```

図 3: MGTP の並列アルゴリズム

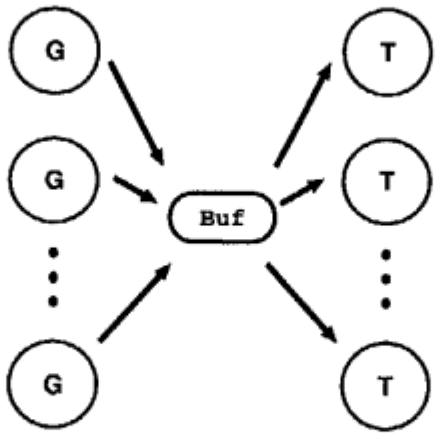

図 4: G/T プロセス

3.3 閉包計算としてみたMGTPアルゴリズム

図1のアルゴリズムがsatを返す時，節集合Sのモデル要素がMDの要素として得られる．モデル拡張規則はもはや適用できず，これ以上のアトムは得られない，という点でMDは閉包になっている．したがって，MGTPのアルゴリズムは，閉包計算のアルゴリズムとして一般化することができる．

様々な計算は，閉包計算を行なっているとみなすことができる[8]．例えば，MGTPの連言照合を一般の超導出，アトム間の包摂テストを節間の包摂テストに置き換えると，超導出手続きが得られる．また，モデル拡張候補を支持集合(set-of-support)と見なし，混合節の連言照合を支持集合導出(set-of-support resolution)に置き換えれば支持集合戦略の証明器が得られる．一方，モデル候補をワーキングメモリ，混合節と負節の集合をルールベース，連言照合をマッチングの計算と考えれば，MGTPはプロダクションシステムに対応する．

このようにMGTPの基本演算を入れ換えるだけで，種々の有用なアルゴリズムを得ることができ，本論文で提案する並列化方式はこれらのアルゴリズムにも適用可能である．

4 AND並列化方式検討

本節では，AND並列化方式を探究する過程で我々がとった設計指針と負荷分散法について述べる．

4.1 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数にかかわらず証明が変わらない証明不変方式とPE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(モデル候補を分割し各PEに分散する)．

(3) マスター有りとマスター無し．

証明不変方式は，PE数によらず同一の証明を得ようとするものである．これを行なうには，アトムの生成順，Bufからのδの取り出し順，包摂テストの順番を固定する必要がある．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が，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，モデル分散方式では，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数が増えるにしたがって通信量が増大してしまう．この生成アトム量は遅延モデル計算の計算量解析[3]によると，μm^2(混合節の前件リテラル数を2，証明終了時のモデル候補数をm，混合節の連言照合成功率をμと仮定)となり，PE数をnとすると$n \times \mu m^2$分の余計な通信が発生する．

マスター・スレーブ方式では，MGTPをスレーブPE上に置き，単にそれらとマスター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．また，マスターが全体の仕事の割当を行なうことにより，負荷の最適配分を容易に実現することができる．

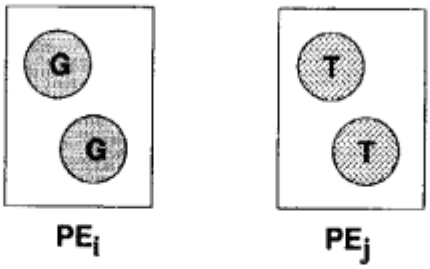

図 5: 機能分散法

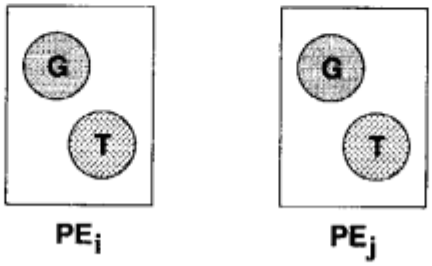

図 6: 機能同居法

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．これに対し，例えば，PE 上に置かれた MGTP をリング状に結合し，各 PE で局所的に負荷配分を行なうようなマスター無しの方式も考えられるが，最適負荷配分が得られるように各 PE を協調的に動作させることは難しくなる．

本 AND 並列化では，(i) 得られた速度向上が並列化によるのか，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，(ii) 実行効率および実現の容易性の両方を追求する，方針を採った．この方針の下に，上記考察を踏まえて，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．

4.2 負荷分散

AND 並列の実現にあたっては，G，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 (図 5) と (2) 各 PE に G，T 二種類のプロセスを重ねて配置する「機能同居」法 (図 6) がある．

(1) の方法では，時々刻々と G，T 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 T 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，軽負荷の PE に G プロセスを配置することが必要であるが，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．

さて，(2) の場合は G，T プロセスを重ねて配置するわけであるから，それぞれの負荷が重ね合わせられ，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．これは一種のプロセスの見かけ上の「動的変身」による動的負荷分散効果

を狙ったものである．この方法ではPE内での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．

我々は，実際に上記二つの方法で並列化実験を行なった．(2) の方が性能はより向上するはずであったが，初期の段階ではなかなか良好な性能を得られなかった．この際に，真価を発揮したのが (1) の方法である．(2) の方法では，G，Tプロセスが重ね合わさっているので，各PEの稼働状況を視覚的に実時間表示するパフォーマンスメータで観察していても，どちらのプロセスが性能劣化の原因となっていのかが分かりにくい．しかし，(1) の方法で走行させると，時間と共に変化するプロセスの負荷比率を観察することができ，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る．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状況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，(2) の方法の性能改善を図ることができた．本論文の評価実験では (2) の方法を採用している．

5 AND 並列化方式

4.1での検討結果を，図3のアルゴリズムに反映させたものが，図7に示すアルゴリズムである．このアルゴリズムは，Gプロセス，Tプロセスに加えてマスター (M) プロセスの三種からなっている．これらのプロセスの論理的結合関係を示したのが図8である．中央のMプロセスを介して上部のGプロセス群と下部のTプロセス群が連結している．

図7では，プロセス間通信を表現するために，Occam風チャネル記法を用いた．$channel!X$ で変数 X の値をチャネル $channel$ に出力し，$channel?X$ でチャネル $channel$ から変数 X に値を入力することを表す．MGC はMプロセスからGプロセスへの，MTC はMプロセスからTプロセスへの，TMC はTプロセスからMプロセスへの通信に用いられるチャネルである．これらの通信を利用することによって，図3で示されていたロック機能と等価な働きを実現することができる[4]．

5.1 動作概要

Mプロセスでは初期化の後，(M1) と (M2)～(M4) が並行実行される．(M1) では各Gプロセスに連言照合の受け持ち範囲 (g で示される) を指示すると共に，生成されたアトムを蓄えるためのバッファ New_g を用意しこれも送付する．New_g は，各Gプロセス毎に用意されるので，この更新にロックは必要ではない ((G4))．

包摂テストは，GプロセスとTプロセスで分離して行なわれる ((G3) と (T1) の条件部)．包摂テストの内，Gプロセスで行なわれるものを局所的包摂テスト (LS)，Tプロセスで行なわれるものを大域的包摂テスト (GS) と呼ぶことにする．ここでは，LSを行なう時点でそのPEに保持されている MD の要素数を m と仮定し，包摂テストの範囲を明示した．例えば，$C\sigma$ is new to $MD[1,\ldots,m]$ で，$\exists\sigma'(C\sigma = MD[i]\sigma' \text{ for } \forall i(1 \leq i \leq m))$ を表している．

生成されLSを通過したアトムは New_g に蓄えられるが，同時にLSの適用範囲を示す m も蓄えられる ((G4))．

Tプロセスは，先ずGプロセスが生成したアトム δ とGSの範囲 $(m,\ldots,s')$ の組をMプロセスから受けとる ((T0))．そして δ が包摂されなければ，$MD[s'+1]$ に δ を代入し，$s'+1$ をMプロセスに送る．δ が包摂されれば，s' をMプロセスに送る．Mプロセスに送った値 (s' または $s'+1$) が δ の次のアトムに必要なGSの範囲を与える．

[4]実際の実装はKL1の未定義変数の送受信を利用して行なわれる．

```
(0)  図1(0)と同じ
(1)  Init: s := k;

process M
(M0) Init: g := 1; g' := 1; b' := 1
(M1) MGC ! {g, New_g}; g := g + 1; goto (M1);
(M2) if New_g'[b'] = empty then begin
(M3)     g' := g' + 1; b' := 1 end
(M4) MTC ! {New_g'[b'], s}; TMC ? s; goto (M2);

process G
(G0) MGC ? {g, New_g}; b := 1;
(G1) foreach A_1, ..., A_n → C do
(G2)     foreach (i_1, ..., i_n) s.t. ∀j(i_j ≤ g) ∧ ∃j(i_j = g) do
(G3)         if ∃σ∀j(MD[i_j]σ = A_jσ ∧    Cσ is new to MD[1, ..., m]) then begin
(G4)             New_g[b] := {Cσ, m}; b := b + 1 end;
(G5)     New_g[b] := empty;

process T
(T0) MTC ? {{δ, m}, s'};
(T1) if δ is new to MD[m + 1, ..., s'] then begin
(T2)     MD[s' + 1] := δ; TMC ! s' + 1;
(T3)     foreach A_1, ..., A_m → false do
(T4)         foreach (i_1, ..., i_m) s.t. ∀j(i_j ≤ s') ∧ ∃j(i_j = s') do
(T5)             if ∃σ'∀j(MD[i_j]σ' = A_jσ') then
(T6)                 return (unsat);
(T7) end else TMC ! s';
```

図 7: MGTP の AND 並列アルゴリズム

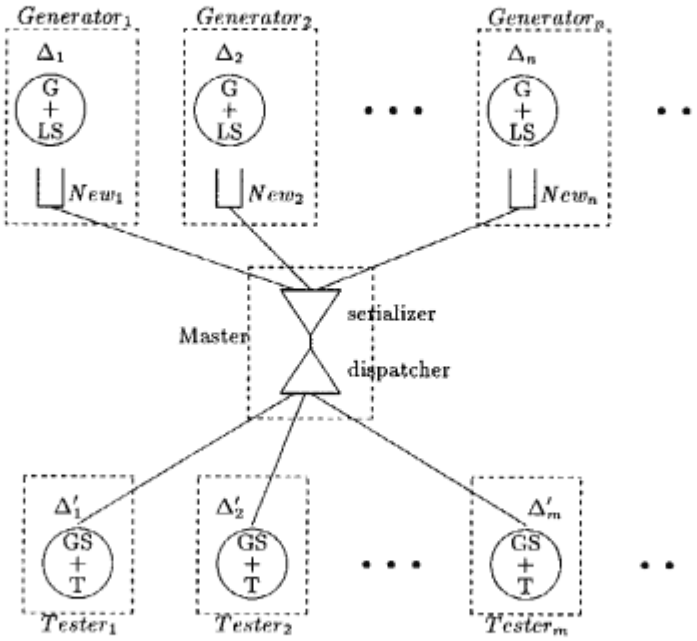

図 8: MGTP の AND 並列化

5.1.1 証明不変と証明可変

M プロセスで，g' を一つずつ増加させることにより，$New_{g'}$ を整列させ，結果として，証明不変方式を実現しているわけである．これを整列させず併合させるようにすれば，証明変動方式になる．また，T プロセス中の s' は，M プロセスによって排他的に指示されるので，これにより MD への書き込みは協調的になる．

5.1.2 負荷分散制御

以上のように，M プロセスの介在によってロックに相当する機能が実現されているが，M プロセスの最も重要な役割は，各 G/T プロセスに負荷を均等に割り当てることである．

G/T プロセスは，割り当てられた仕事が終了すると次の仕事をもらうために M プロセスに要求を出す ((GO)/(TO))．M プロセスは，1) 要求に先着順に応答し，2) 各 G/T プロセスが同じ仕事をしないように排他的に仕事を割り当てる．この際に，M プロセスは，G プロセスが生成したアトム数と T プロセスが検査したアトム数の差 ($New_g[b]$ と $New_{g'}[b']$ の差) を一定に保つことにより，G プロセスの過剰生成を抑制する．

これは，いわゆるバッファリング[5]であり，この差がある一定数を越えると (バッファが溢れそうになる) と G プロセスからの要求を一時保留し，T プロセスの要求のみを受け付け，逆にバッファが空に近づくと，G プロセスからの要求を優先的に受け付ける．この機構によって間接的に遅延モデル生成が実現され，G/T プロセス間の負荷の均衡が計られる．

またこのような機構は，KL1 のプロセス間通信の技法と節優先度指定プラグマ (alternatively) を利用することにより，簡単に実現することができる．

[5]各 New_g がバッファの働きをする．

5.2 実装上の留意点

本並列化方式には原理的に二つの逐次性があり，並列性能を引き出す上での阻害要因となっている．一つは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．

5.2.1 包摂テスト

新たに生成されたアトムの前方包摂テストを完全に行なうには，その生成順を保持して，以前に生成されたアトム集合に対して包摂テストを行なう必要があるため逐次性が生じる．この逐次性の低減を図るため，包摂テストをLSとGSに分離した．

モデル共有方式を採用しているので，各生成アトムに対するLSは他のPEに影響を与えることなく，各PE内でそれぞれ独立に行なうことができる．但し，LSの範囲は広ければ広いほど(図7の(G3)中のmが大きければ大きいほど)，アトムが包摂される可能性が高まり，結果として通信とGSの負荷が減るので好ましい．従って，あるPEでのMDの更新結果は，直ちに他の全PEに伝搬するような実装が必要である．

各PEで独立に行なうことのできるLSに対し，GSには逐次性が残っている．これは，直前のアトムの包摂テストが終了しないと，次のアトムの包摂テストが終了できないことに起因する．図7でいうと，(T1)でのδ is new to $MD[m+1,\ldots,s']$の判定には，$MD[m+1],\ldots,MD[s']$の値が全て確定，つまりそれらの包摂テストが全て終了している必要がある．

このGSの逐次性を低減するために，LS完了後のアトムを保持するための大域的な配列TMD(仮積みリストと呼ぶ)を用意する．これは，図3で使用しているBufと実際には全く同じである．TMDの要素の内GSで包摂されなかったアトムが，MDの要素となるので，MDはTMDの部分集合となる．

TMDを使った場合の手続きは，図7を，次のように変更すればよい．TMDは，MDと同様な初期化手続きがなされ，$ts := k+1$と初期化がなされているとする．

(1) (M4)を次のように変更する: $MTC\,!\,\{New_{g'}[b'],s,ts\}; ts := ts+1; TMC\,?\,s;$

(2) (T0)を次のように変更する: $MTC\,?\,\{\{\delta,m\},s',ts'\};$

(3) (T0)と(T1)の間に以下を挿入する: $TMD[ts'] := \delta$

(4) (T1)の条件部を次のように変更する: δ is new to $TMD[(m+1)^\flat,\ldots,ts'-1]$
ここで，$()^\flat$は，MDのインデックスからTMDのインデックスへの関数で$MD[m] = TMD[(m)^\flat]$を満たす．

(3)によって，GSを始める前にアトムを仮積みリスト(TMD)に積んでおくので，このアトムのGSの終了を，他のアトムのGSが待つことはない．但し，仮積みリストには，包摂されるかも知れないアトムも置かれる可能性があり，このようなアトムに対する包摂テストは，仮積みリストを設けない場合には行なわれないので，冗長である．仮積みリストによる逐次性の削減は，この冗長計算と引き替えに得られるものである．

5.2.2 証明不変

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G_1とG_2が並列に動いている時，G_1の生成アトム集合New_1とG_2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．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 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，T プロセスが動けない状態になってしまう．G_1 の実行が滞っていたとしても，G_2，G_3 ... 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．

そこで上記の逐次性を低減するために，G プロセスの粒度を細分化することにした．具体的には，これまで G プロセスが行なっていた仕事の単位 (図 7 において，一つのアトム $MD[g]$ に対する (G1) のループの処理) をさらに $1/k$ に均等分割し，これを仕事の単位とする．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った．

6 PIM 上での実験・評価

以上のような実行モデルに基づき 4 種の並列版を実装した．それらは，1) 局所的包摂テスト (LS) にインデックスによるアトム検索を行なう弁別木 [9] を用いるか否か[6]，2) 大域的包摂テスト (GS) のための仮積みリストを保持するか否か，の 2 × 2 の計 4 通りである．

6.1 実験

並列化効果を評価するために，PIM/m-256 システム [7] 上で，分離規則に関する問題 [5] の内の 23 題を用いて実験を行なった．

PIM/m-256 システムは，分散メモリ構造の大規模 MIMD 計算機であり，256 台の PE が二次元格子状に高速ネットワークで接続されている．各 PE は，タグ操作やデータ型判定機構，参照ポインタを自動的に手繰りよせる機構など，KL1 をはじめとする記号処理言語の実現に適した機構を備える．

分離規則に関する問題は ¬，≡，∨ や群の演算子等に関する一つの分離規則 (detachment rule (modus ponens)) といくつかの公理および解きたい定理 (負節で表現) からなり，すべて以下のようにホーン節のみの集合で記述される．

 # 60 (i : implication / n : negation)
$p(X),\ p(i(X,Y)) \rightarrow p(Y).$
$true \rightarrow p(i(X,i(Y,X))).$
$true \rightarrow p(i(i(X,Y),i(i(Y,Z),i(X,Z)))).$
$true \rightarrow p(i(i(i(X,Y),Y),i(i(Y,X),X))).$
$true \rightarrow p(i(i(n(X),n(Y)),i(Y,X))).$
$p(i(i(a,b),i(n(b),n(a)))) \rightarrow false.$

最初の節が分離規則を表す混合節で，混合節はこれ一つである．それに続く正節で公理を表し，最後の負節が解きたい定理の否定である．[5] には，このような問題が 112 題掲載されており，#60 というのはその通し番号である．

実験に用いたのはこの内の 23 題で，問題を証明時間の長短により二つのグループに分けた．32PE を用いた場合にいずれの方式でも 500 秒以内に証明できた 11 題からなる第一グループ[7]と，500 秒以上を要した 12 題からなる第二グループ[8]である．

表 1 は各方式において 32，64，128，256PE システムを用いた時の，第一グループ (500 ≤) と第二グループ (500 >) の問題の一題あたりの平均証明時間と 32PE システムの証明速度を 1

[6] 用いない場合は，線形探査を行なう．GS はいずれの場合も線形探査

[7] #3，#14，#19，#42，#60，#75，#78，#81，#98，#103，#106

[8] #15，#21，#22，#40，#43，#44，#49，#60，#62，#67，#79，#82

表 1: 各並列化方式の性能と速度向上比

方式		問題	実行時間 (秒: 一題平均)/ 速度向上比 (32PE = 1)			
1†	2‡		32 PE	64 PE	128 PE	256 PE
無	無	500≤	196/1.00	113/1.73	85/2.30	142/1.38
		500>	28157/1.00	14269/1.97	7600/3.70	4151/6.78
無	有	500≤	201/1.00	115/1.75	76/2.65	84/2.40
		500>	29852/1.00	14915/2.00	7979/3.74	4424/6.75
有	無	500≤	112/1.00	65/1.73	85/1.32	211/0.53
		500>	4542/1.00	2241/2.03	1178/3.85	1113/4.08
有	有	500≤	116/1.00	68/1.71	47/2.51	65/1.79
		500>	4597/1.00	2287/2.01	1212/3.79	756/6.09

†弁別木有り / 無し ‡仮積みリスト有り / 無し

表 2: 並列実行性能†(#60 と #67)

問題	実行時間 (立ち上がり時間)(秒)/ 速度向上比 (32PE = 1) LS 生存率 /GS 生存率‡(%)			
	32 PE	64 PE	128 PE	256 PE
#60	5061(1)/1.00	2619(3)/1.93	1440(10)/3.51	1051(70)/4.82
	3.70/90.68	3.85/87.19	4.07/82.30	4.37/76.36
#67	1770(1)/1.00	886(4)/2.00	480(12)/3.69	340(40)/5.20
	0.12/43.93	0.14/37.97	0.16/33.56	0.19/27.69

†弁別木有り・仮積みリスト有り方式
‡LS(/GS) にかけられたアトムの内，包摂されなかったアトム数の割合

とした場合の証明速度比を示している．例えば，256 PE システムの速度比が 8 ならば線形の速度向上が得られたことになる．

表 2は，第二グループの #60 と #67 に関するより詳しい計測結果を示している．表のデータは，弁別木有り・仮積みリスト有り方式によるもので，PE 数増加にともなう速度向上の他に，包摂テストによるアトムの生存率 (包摂テストにかけられたアトムの内，包摂されなかったものの比率)，立ち上がり時間 (証明開始から全 PE がフル稼働になるまでの時間)，を示している．

6.2 考察

以下に示すような各項目について，並列実行効果の観点から計測結果を考察する．

6.2.1 第一グループと第二グループ

第一グループの問題に対しては 64~128PE 程度までは，並列実行効果を認めることができるが，256PE では明らかにタスクの分割損が現れている．表 2で示している立ち上がり時間の

傾向はほぼ全問題に共通しており[9]，32PE及び64PEでは，数秒で全PEがフル稼働状態になるが，128PEで10秒程度，256PEでは数十秒を要する．第一グループの問題は証明時間が短いため，全PEがフル稼働するまでの間に証明が終ってしまうのである．これらは，256PEを投入するまでもない問題であるともいえるが，フル稼働するまでの立ち上がりの時間をさらに短縮することによる性能改善は，今後の課題である．

以降の議論は，専ら第二グループの問題の証明時間に関して行なう．

6.2.2 弁別木有りと無し

絶対性能は，仮積みリストの有無に関わらず弁別木有りの方がかなり優位である．このことからも包摂テストの性能向上が定理証明器にとって重要な検討課題であることがわかる．ただ速度向上比は，弁別木無しの方が優れている．この理由は無しの場合，LSに費やす処理時間，つまりLSの粒度が大きくなり，その結果GSの逐次性が相対的に小さくなったためであると考えられる．

このことは，もし連言照合やLSの速度がより増せば，GSの逐次性がより顕著になり台数効果が得られにくくなることを示唆している．

6.2.3 仮積みリスト有りと無し

弁別木無しの場合は，仮積みリストのオーバヘッドが少し現れており，仮積みリスト無しの方の性能が若干優れている．仮積みリストは，GSの逐次性を低減するために導入されたものであるので，GSの逐次性が相対的に小さくなる弁別木無しでは，その効果が現れずオーバヘッドの方が顕在化したからである．弁別木有りの場合では，32，64，128PEと仮積みリストの効果もオーバヘッドもほとんど見られないが，256PEの場合には，その効果がオーバヘッドに勝り，性能が3割以上改善されている．

これは，多少のオーバヘッドと引き替えてでも，逐次ボトルネックの解消を計る方が，並列実行の場合には有効であることを示している．

6.2.4 アトムの包摂テストによる生存率

表2より，LSによるアトムの生存率が低い問題#67の方が並列性能を得やすいことが分かる．LSを通過したアトムは，GSを行なうために全PEにコピーされる．つまり，#60のようにLSによる生存率が高い問題ほど，通信の頻度が増し，これが並列実行のオーバヘッドとなる．

LSによるアトムの生存率は，使用PE数が増えるに従い，増加していることも分かる．LSは，各PE毎にコピーされ局所的に保持しているモデル候補及びモデル拡張候補に対して行なわれる．PE数が増えるに従い，これらの局所的データの更新の遅れが大きくなり(コピーの量が増えるため)，結果としてLSの範囲が狭まり，生存率が増加しているものと考えられる．投入PE数に比例して，実行性能が向上しない原因の一つに，この生存率の増加があげられる．

また，データとしては陽に現れていないが，一括ゴミ集め(GC)の影響も無視できない．PEの稼働状況を観察していると，GCの直後には，数秒，場合によっては十数秒の間稼働率が著しく低下する．この現象は，LSによるアトムの生存率が高い問題ほど顕著になる．これは，LSを通過したアトムの通信が増えるとPE間にまたがるゴミも増え，その回収に伴う影響によるものと思われる．これも生存率の高い問題が低い問題に比して，台数効果の劣化する理由の一つである．

---

[9] PE数をnとすると$O(n^2)$時間かかっている．これは，他のPEでモデル拡張候補に登録されたアトムを自PEにコピーする際の待ち時間が1アトムあたり$O(n)$となり，他のPEで登録された$n-1$個のアトムをコピーするのに$O(n^2)$時間待たされるためと考えられる．

6.2.5 マスターの負荷

実行時のマスタープロセスの稼働状況を観察している限り，256PE 使用まではマスターネックの問題は発生しておらず，マスターの負荷は軽いといえる．現方式のままでも，G プロセスの粒度を粗くしてマスターの負荷を軽減することにより，1000PE 程度までは対処可能であると思われる．

しかしながら，マスターネックの解消は避けて通れない問題であり，1000 台規模を超える場合には，マスターの分散化あるいは階層化といった対処が必要になるであろう．

6.2.6 メモリ消費量

PIM/m 1PE 当たりのユーザが利用可能なメモリ量は30MB 程度である．分離の規則に関する問題 112 題の内，10 題ほど証明できていないが，これらの問題は，256PE 使用で 4 時間から 12 時間程の走行で利用可能メモリの 9 割を消費する．また，メモリが残り 1 割を切るとGC 頻発のため，ほとんどの CPU 時間は，GC のために費やされるようになる．

現在，このメモリ問題を解決するためモデル分散方式や不要なアトムの削除戦略について検討中である．

7 むすび

モデル生成型定理証明系 MGTP の AND 並列化方式を提案し，その性能評価を行なった．その結果，十分大きな問題 (第二グループの問題) については，256PE を用いると，32PE を使用した場合と比較して，6 倍強の速度向上を実現した．64PE 使用程度までは線形台数効果が得られているので，結局，256PE で 200 倍程度の台数効果が得られたことになる．

MGTP の単一化プログラムの速度は，C で記述されている汎用定理証明系 OTTER[6] の速度と比べて約 3 倍ほど低速 (PIM/m-1PE と SPARC-2 との比較) である．システム全体の性能は，単一化プログラムの性能にほぼ比例するものとすれば，256PE 使用で，Unix マシン (SPARC-2) の約 80 倍の速度が得られる計算になる．

第二グループの問題の内 6 題[10]は，OTTER で様々な戦略をもってしても証明できず，並列実行により初めて機械的に証明することのできた問題である．このことも，自動定理証明における並列計算の有効性を示している．

現在の AND 並列化方式は包摂テストの逐次性の問題は完全には解決しておらず，また，メモリ・スケーラビリティの点で限界がある．今後，アトム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．メモリ・スケーラビリティについては，モデル分散・マスター無しの並列化方式を実現する必要がある．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 統合方式を確立していく予定である．

謝辞

本研究に関して，有益な助言を頂いた三菱電機中央研究所・藤田博氏，計測の労をとって頂いた九州ジェーピーエー・山鹿英樹氏に感謝する．また，本研究の機会を頂いた ICOT の渕一博前研究所長 (現東大教授)，内田俊一研究所長に深く感謝する．

[10] #15, #22, #40, #43, #44, #49

参考文献

[1] Chang, C-L. and Lee, R. C-T.: Symbolic Logic and Mechanical Theorem Proving. Academic Press, 1973.

[2] Fujita, H. and Hasegawa, R.: A Model Generation Theorem Prover in KL1 Using Ramified-Stack Algorithm. In *Proc. 8th ICLP*, pages 535 548, Paris, 1991. also in ICOT TR-606 1990.

[3] Hasegawa, R., Koshimura, M. and Fujita, H.: Lazy Model Generation for Improving the Efficiency of Forward Reasoning Theorem Provers. In *Proc. Int. Workshop on Automated Reasoning*, pages 221–238, Beijing, 1992. also in ICOT TR-751 1992.

[4] Manthey, R. and Bry, F.: SATCHMO: a Theorem Prover implemented in Prolog. In *Proc. 9th Int. Conf. on Automated Deduction*, Argonne Illinois, 1988.

[5] McCune, W. W. and Wos, L.: Experiments in Automated Deduction with Condensed Detachment. In *Proc. 11th Int. Conf. on Automated Deduction*, pages 209–223, Saratoga Springs, NY, 1992.

[6] McCune, W. W: OTTER 2.0 Users Guide. Tech. Report ANL-90/9, Argonne National Laboratory, Argonne IL, 1990.

[7] Nakashima, H., Nakajima, K., Kondoh, S., Takeda, Y., Inamura, Y., Onishi, S. and Kasuda, K.: Architecture and Implementation of PIM/m. In *Proc. Int. Conf. on Fifth Generation Computer Systems 1992*, pages 425–435, Tokyo, 1992.

[8] Slaney, J. K. and Lusk, E. L.: Parallelizing the Closure Computation in Automated Deduction. In *Proc. 10th Int. Conf. on Automated Deduction*, pages 28–39, Springer-Verlag, 1988.

[9] Stickel, M.: The path-indexing method for indexing terms. *Technical Note* 473, Artificial Intelligence Center, SRI International, Menlo Park, CA, October 1989.

[10] Suttner, C. B. and Schumann, J.: Parallel Automated Theorem Proving. In Kanal, L., Kumar, V., Kitano, H. and Suttner, C.B., editors, *Parallel Processing for Artificial Intelligence*, Elsevier, 1994.

[11] Ueda, K. and Chikayama, T.: Design of the Kernel Language for the Parallel Inference Machine. *Comput. J.*, December 1990.

[12] 藤田 正幸, 桑野 文洋: MGTP による有限代数の新事実の発見. JSPP'93 論文集, 1993.