ICOT Technical Report: TR-575

TR-575

共有メモリマルチプロセッサにおける KL1言語の並列実行方式 負荷分散とユニフィケーション—

今井　明，後藤厚宏，堂前慶之

July, 1990

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

共有メモリマルチプロセッサにおけるKL1言語の並列実行方式
– 負荷分散とユニフィケーション –

KL1 Parallel Execution on Shared Memory Multi-Processor
– Load Balancing and Unification –

今井 明　後藤 厚宏　堂前 慶之
Akira IMAI　Atsuhiro GOTO　Yoshiyuki DOUMAE
新世代コンピュータ技術開発機構　富士通ソーシアルサイエンスラボラトリ
Institute for New Generation Computer Technology (ICOT)　Fujitsu Social Science Laboratory Ltd.

概要

本稿では，並列論理型言語 KL1 を並列推論マシン PIM のクラスタのような共有メモリ結合マルチプロセッサ上での実行方式の一例について述べる．まず，負荷分散手法として KL1 のゴールを最小単位とし，要求駆動により自動負荷分散方式を実現することで，高負荷時の処理を高速化する方式を述べる．また，論理型言語処理系で基本的な処理であるユニフィケーション処理では，排他制御を Compare & Swap を用いて共有変数のロック期間を短くする方式や，通常の KL1 ゴールを用いることで複雑な処理を単純化する方式について述べる．

Abstract

In this paper, we describe one scheme of parallel execution for concurrent logic programming language KL1 on shared memory multi-processor such as PIM cluster. We focus on two topics. One is an automatic load balancing scheme based on demand driven strategy which accelerates when high load average. The other is an implementation technique of unification which is one of the basic operations in logic programming language processing. This technique reduces the term of locking shared variable using "Compare & Swap" primitive and simplifies using KL1 goal reduction.

1　はじめに

ICOT では，並列論理型言語 KL1 を高速に実行する並列推論マシン PIM[1] の研究開発を行っている．PIM は，通信コストを抑えるために 8 台程度のプロセッシングエレメント (PE) を共有メモリで密に接続したクラスタを，大規模並列マシンとするために 専用の高速ネットワークで疎に結合する階層構成とする．

PIM の上で動く高度な応用システムを KL1 のような高級な言語で記述することは，並列処理特有の同期 / 通信といった複雑な処理をできるだけ意識しないで解法アルゴリズムを記述できる点で非常に有効である．しかし，このような高級な並列言語の処理系を構築する際には，同期や排他制御を常に考慮する必要があり，言語が高機能になればなるほどこれらは複雑化する恐れがある．

我々は，PIM 用の KL1 言語処理系となる VPIM[2] のクラスタ内処理方式の設計に当たって，高速なプロセッサ間通信が可能であるという共有メモリ結合の利点を生かしながら，共有データアクセスの競合を避けるとともに，複雑になりがちな排他制御処理を単純化することに特に注意した．

本稿では，KL1 言語処理系の基本処理である負荷分散とユニフィケーションを例にして，この方針に基づいて設計した詳細処理方式を報告する．

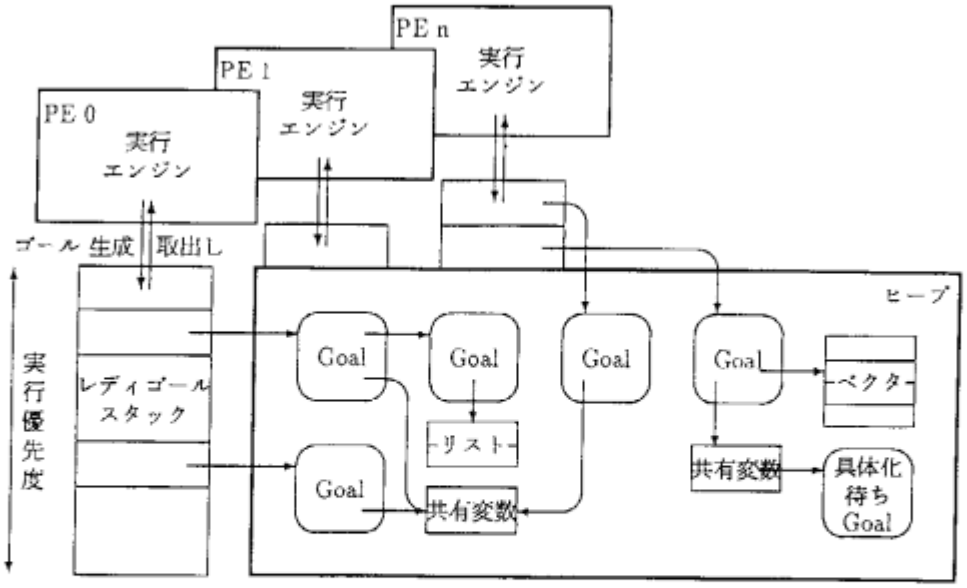


図 1: KL1 の実行イメージ

2 共有メモリマルチプロセッサでの KL1 並列実行方式の概要

2.1 KL1 のシンタックス

並列論理型言語 KL1 は, Flat-GHC[3] を基に ICOT で設計した言語であり, KL1 プログラムは次のようなシンタックスを持つ節の集合として表される.

$$H :- G_1, \cdots, G_m \mid B_1, \cdots, B_n. \quad (m \geq 0, n \geq 0)$$

ここで, H, G_i, B_i をそれぞれ, クローズヘッド, ガードゴール, ボディゴールと呼ぶ. '|' はコミットメントオペレータと呼び, これ以前の部分を受動部, これ以後の部分を能動部と呼ぶ. 能動部中のゴール $B_1 \cdots B_n$ 間には実行順序の概念はなく, すべて並列に実行できる. これらの同期はゴール間で共有した変数を受動部で待ち合わせることで行う.

2.2 設計方針

アドレス空間を共有した共有メモリマルチプロセッサでは, プロセッサ間通信を共有メモリへの Read/Write 操作によって高速に行うことができる.

このようなマルチプロセッサ上で動作する言語処理システムの設計に当たっては,

排他制御による処理の直列化で並列性を下げない工夫

「際どい領域」に入って行わなければならない処理を少なくし, できるだけ共有データへのアクセス競合が起きないようにすること

排他制御自体を不要とする工夫

データを置く論理的な領域を言語処理システムのメモリアクセス特性に応じて分離し, 他のプロセッサが直接アクセスできないようにすることで, そもそも排他制御を不要とすること

が重要となる.

2.3 KL1 の実行イメージ

KL1 の実行イメージを, 図 1 に示す.

KL1 ゴールは, ヒープ (変数領域) の変数セルへのポインタを共有し, それをユニフィケーションという操作で具体化することで同期や通信を行う.

ゴールは引数, コードへのポインタなどの実行環境を記録したゴールレコードで表現する. ゴールレコードも通常の変数セルやリストと同様にヒープ中に割り付ける. 実行可能なゴールのゴールレコードは, そのプライオリティ (実行優先度) に応じて, レディゴールスタック中からリンクを張り, 最高位のプライオリティから順に一つずつ取り出してリダクションを行う. リダクションの結果, 新しいゴールを生成する時には, ゴールレコードを割り付けて実行環境を記録した後に, レディゴールスタックに入れる.

また変数が未定義であるために実行できないゴールは, 変数が具体化された時に効率良く再開できるように, 未定義変数からリンクを張る (フック操作と呼び, これにより同期機構を実現する).

また, 記号処理言語に必須のガーベジコレクション (GC) 処理も, KL1 のような副作用のない言語では特に重要で, この高速化のために即時形と一括形を組み合わせることで両者の欠点を補完する. また, それらはいずれも並列に動作する方式を設計している [4].

3 負荷分散方式

3.1 KL1 における負荷分散の単位

前述のように，KL1 のボディゴール間には実行順序の概念がなく，並列に実行してよい．すなわち，KL1 の実行では，ゴールを単位とした負荷分散を行うことができる．これより細かい単位，即ちユニフィケーションのような処理を並列に行うことも可能であるが [5]，KL1 のユニフィケーションでは並列化して台数効果を期待できるほど複雑なパターンマッチングは少なく，並列化のためのオーバーヘッドも大きくなることから，高速化のための負荷分散単位としては現実的ではない[1]．

PIM のクラスタ間のようにメモリを共有しない並列マシンでは，プロセッサ間通信のコストが共有メモリに比べて格段に大きいため，負荷分散戦略は KL1 の負荷分散プラグマを用いた応用問題の解法の一部となる [6]．本方式は通信コストの比較的小さいクラスタ内で負荷を動的に融通することで，クラスタがマルチプロセッサであることを意識させないことを狙ったものである．

3.2 要求駆動による負荷分散方式

実行可能なゴールレコードを保持したレディゴールスタックは，PE 毎に持つ．そして，リダクションの結果子ゴールを生成した場合，そのゴールレコードは，これを生成した PE のレディゴールスタックに入れる．そのため，通常は PE 間でゴールの融通を行わない．

リダクションが進むと，

- 実行すべきゴールがなくなり，アイドルになる
- クラスタ内で実行中のゴールのプライオリティの平均値より低いゴールの実行が続く

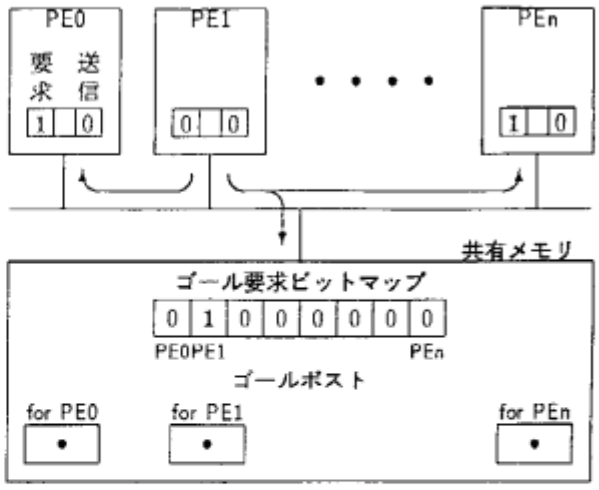
という状態が起こり得るが，このような状態になった PE は他の PE に対してゴール送信要求を出し，自分のゴールポストにゴールを贈いでもらうことで負荷分散を行う [7]．

以下，PE1 がゴール要求を出し，PE0 がこの要求に応えることを例に，この処理の詳細を述べる．なお，以下に現れる“ゴール要求ビット”,“ゴール送信ビット”は PE 間で共有されるデータであるため，共有メモリ中に置いても良いが，PIM では放送型のスリットチェックレジスタという特殊レジスタを用いて共有バストラフィックを軽減する [8]．

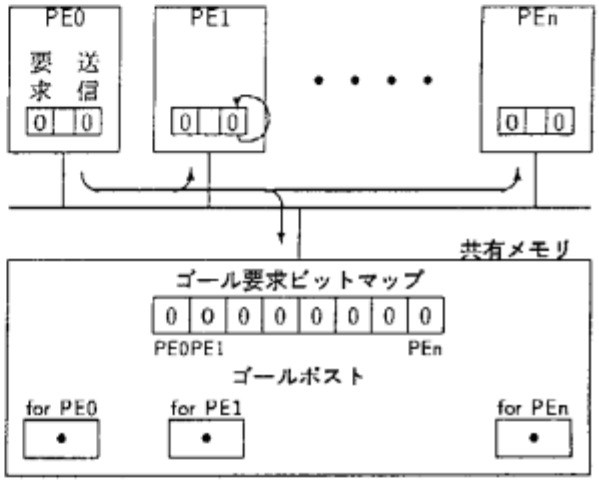
3.2.1 ゴール要求処理

ゴール要求は以下のような手順で行う．

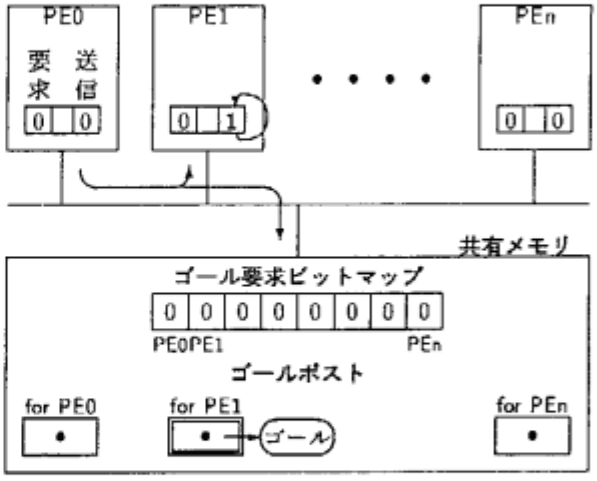
[1]後述する構造体同士のユニフィケーションは並列に行うことがあり得るが，並列処理による高速化を狙ったものではなく，処理の単純化を狙ったものである．



a) PE1 のゴール要求処理



b) PE0 がポーリングでゴール要求に気付く



c) PE0 が PE1 にゴールを送出

図 2: 要求駆動による負荷分散方式

1. ゴール要求の有無を示したビットマップの自分のビットを立てると同時に，クラスタ内の他の PE の“ゴール要求ビット”を立てる (図 2a).

2. 他の PE でこのイベントが処理され，自分の“ゴール送信ビット”が立てられるまで，このビットをポーリングして待つ.

3.2.2 要求を受信したプロセッサのゴール送信処理

リダクションを終え，次のリダクションを開始するタイミングで，“ゴール要求ビット”が立っていることに気付いた PE は，ゴールを分配するために次のような処理を行う.

1. ゴール要求ビットマップのパターンを読み込むと同時に，他の PE の“ゴール要求ビット”を下ろす (図 2b).

2. 自分のレディゴールスタックの中からプライオリティの一番高いゴールを一つ取り出し，ゴール要求を出している PE のゴールポストに繋ぐと同時に，その PE の“ゴール送信ビット”を立てる (図 2c).

なお，ゴール要求をしていた PE は 1 台とは限らないので，ビットマップ中のビットの立っていた PE 全てに対して，2 の処理を行う必要がある．ここで，2 の処理を続けているうちに，送信可能なゴールがなくなってしまう場合があり得る．この場合は，ビットマップが立っていたにもかかわらず送信できなかった PE の代わりにゴール要求を出す.

3.2.3 ゴールを送信されたプロセッサの処理

他の PE によりゴールがポストに繋がれ，“ゴール送信ビット”が立てられると，これをポーリングして待っていた PE はゴールを送信されたことに気付く.

そこで，ゴールポストから送信されたゴールを取り出して，そのプライオリティに応じたレディゴールスタックの位置にゴールを繋ぐ．なお，ゴールは複数繋がっている可能性があるので，すべてのゴールをレディゴールスタックに繋ぐ処理を行う必要がある.

3.3 考察

上述の負荷分散方式について考察する.

共有メモリマルチプロセッサ上の Parlog システム [9] では，アイドル状態[2]になると，自ら他 PE のレディゴールスタックからゴールを取る.

実用的な大規模プログラムでは，実行可能なゴール数がプロセッサ数に対して十分大きいと考えられるので，本負荷分散方式は，

- ゴールの出し入れがローカルなゴールスタックを対象にするため，排他制御が不要
- 要求がない時のオーバーヘッドは，1 リダクションに要求ビットマップのテストを行うことだけである

などの特長を持つ．また，刺り込みで処理する場合の

- 割り込み処理の前後に，レジスタの退避 / 復帰の処理が入る
- 全ての PE に割り込むと，全ての PE で割り込み処理の手間が必要で，かつ必要以上のゴールが送信される
- 特定の PE だけに割り込みを行うと，その PE がゴール要求中の場合の処理が更に複雑になる

などの問題を要求側 PE(先の例では PE0) のポーリングによって回避している.

ただし，プライオリティの変化が頻繁に起こり，かつプライオリティを厳密に守ることで効率良く探索木の枝刈りが可能となるプログラムでは，Parlog 方式や，レディゴールスタックを共有するなどの方式が有効であることも考えられ，今後様々な方式を評価してゆきたい.

---

boolean Compare & Swap (ptr, old, new)

ptr の指す先が old に等しい時，new を ptr の指す所に書き return true

ptr の指す先が old と異なる時，その内容を old に代入し return false

図 3: Compare & Swap

4 ユニフィケーションの処理方式

ユーザが記述した KL1 プログラムは，ゴール間で共有する変数に対するユニフィケーションという操作でプロセス間の同期 / 通信を行う．前章で述べたように，ゴールは負荷分散により他のプロセッサに投げられるため，ユニフィケーションで共有変数の値を書き換える操作では排他制御が必要になる．この排他制御は，共有変数に対するロック期間が短い Compare & Swap プリミティブを利用する (図 3).

4.1 受動部におけるユニフィケーション

KL1 の受動部におけるユニフィケーションでは，未定義の変数に値を代入する操作は許されておらず，具体化した変数の値を参照する操作のみが許されている．KL1 変数の単一代入の性質により，一度値の具体化した変数は書き変わる

[2]プライオリティが言語仕様としてサポートされていないため，アイドルになった状態だけである.

可能性がないため，具体化した変数の読み出しには排他制御は不要である．

ただし，読み出し対象の変数が未定義の場合で，この変数の具体化後でなければ実行できない場合は，前述のフック操作を行う[3]．これは，共有変数に対する書き換え操作となるので，Compare & Swap によって行う．

4.2 能動部におけるユニフィケーション

4.2.1 共有メモリ結合であるための追加処理

共有メモリ結合マルチプロセッサ上の KL1 処理系では，単一プロセッサ上あるいは疎結合マルチプロセッサ上の処理系と比較して，次のような処理の追加が必要である．

未定義変数を具体化する時の排他制御

能動部におけるユニフィケーションでは，未定義変数の具体化を行うことができるが，この時同時に他の PE が具体化を行おうとしている (または，前述のようなフック操作を行おうとしている) 場合があり，排他制御が必要となる．

未定義変数同士のアドレス比較

未定義変数同士のユニフィケーションでは，2 つの未定義変数が同一であることを示すために，一方の未定義変数セルからもう一方の未定義変数セルに対してリンクを張る操作を行う．このとき，同時に他の PE が全く同じ 2 つの未定義変数同士のユニフィケーションを行う可能性があるため，無作為にリンクを張るとループができてしまう．そこで，未定義変数セルの置かれたアドレスを比較して

> アドレスが上位のセルから下位のセルに向けてリンクを張る

という取り決めにより，この問題を解決する．

4.2.2 排他制御を単純化するための工夫

排他制御を単純化するために導入した手法を述べる．

Write Mode を仮定したユニフィケーション

```
p(X) :- true | X = [a|b].
```

のような能動部におけるユニフィケーションでは，Read Mode と Write Mode の 2 通りがある．

モード	X の状態	処理内容
Write	未定義	X を [a|b] で具体化する
Read	具体化済み	X と [a|b] の比較を行う

[3]論理的には未定義の変数であることには変わりなく，厳密に言えばこの処理はユニフィケーションではない．

そこで，ユニフィケーションの方法としては，

- Write Mode を仮定する方式 [10]
  ユニフィケーションに先だって，CAR が a, CDR が b のリストセルを割り付け，これを X とユニファイする．
- Read Mode を仮定する方式
  X と リストの枠をまずユニファイし，その後に，CAR を a, CDR を b とユニファイする．

が考えられる．

前者は X が既に具体化済みの場合に不要なリストセルを割り付けることになるが，後者は，Write Mode の場合に X を具体化する PE が CAR, CDR をユニファイし終わるまでリストセルをロックし続ける必要があり，Compare & Swap を用いることができない．

殆どの KL1 プログラムでは 能動部のユニフィケーションは Write Mode であるため，ロック期間が短く単純な前者を選択する．

KL1 ゴールによる再試行

生起頻度の低い事象が発生した時の処理を単純化するために，通常の KL1 ゴールによる再試行という手法を用いる．

- Compare & Swap に失敗した時

未定義変数への書き込みを Compare & Swap によって行おうとし，これに失敗した場合 (他の PE によって書き換えられていた場合)，通常はユニフィケーション処理の先頭に戻って Compare & Swap をやり直す必要がある．この処理を単純化するために，KL1 の能動部におけるユニフィケーションが，ゴールの実行と同様に実行順序の概念がないことを利用した次のような方法で実現する．すなわち，

```
unify_retry(X, Y) :- true | X = Y.
```

というプログラムを予め組み込んでおき，X に Compare & Swap に失敗した未定義変数へのポインタ，Y に Compare & Swap 成功時に書き換える値を載せて，この unify_retry ゴールを通常のユーザゴールと同じレディゴールスタックに入れてスケジュールすることで，処理を単純化する．

- 構造体同士のユニフィケーション

構造体同士のユニフィケーションでは，それらの要素をすべてユニフィケーション可能であることを調べる必要がある．このような処理では，スタックを用いることが一般的であるが，処理が複雑になる．そこで，リストやベクタ同士のユニフィケーションが次のような KL1 ゴールで表現できることを利用して，処理を単純化する．すなわち，

```
list_unifier([X1|X2], [Y1|Y2]) :- true |
    X1 = Y1, X2 = Y2.
vect_unifier(0, V1, V2) :- true | true.
vect_unifier(N, V1, V2) :- N > 0,
    N1 := N - 1,
    vector_element(V1, N1, Elm1, NV1),
    vector_element(V2, N1, Elm2, NV2),
    Elm1 = Elm2,
    dcode_vect_unifier(N1, NV1, NV2).
```

というプログラムを予め組み込んでおき，ユニフィケーション対象の2引数を載せて，通常のユーザゴールと同じレディゴールスタックに入れ，スケジュールする．

専用のスタックを用いる場合に比べ，このKL1ゴールを用いた処理方式では

○ ユーザゴール同様にスケジュールすれば良く，特別なスケジューリングを考える必要がない．

○ スタックの物理的サイズに仕様が規定されない．

× 処理が遅い

という特徴がある．ただしKL1では，そもそも構造体同士のユニフィケーションを能動部で行うことが稀であるため，処理の遅さは特に問題にならないと考えられる．

5 おわりに

KL1処理系を，共有メモリ結合マルチプロセッサ上で実現するための，要求駆動による負荷分散方式を示し，また複雑になりがちなユニフィケーションをCompare & SwapやKL1ゴールを用いて単純化する方式について述べた．

今後，更に静的解析によってゴール間の依存関係を調べるようなアプローチも考えてみたい．すなわち，これにより送出候補ゴール間の優先度をつけてプロセッサ間通信量を削減したり，他のプロセッサに書き換えられる可能性のない場合に排他制御を行わないで変数の具体化を行うような最適化が見込めるが，詳細は未検討である．

謝辞

日頃，貴重なご意見ご討論を頂くICOT第1研究室瀧和男室長をはじめとするPIM / Multi-PSI開発スタッフに感謝します．特に，本方式の基本設計を行った，沖電気総合システム研究所の佐藤正俊氏，日立製作所中央研究所の中川貴之氏に感謝します．最後に，本研究の機会を与えて頂いた，ICOT渕一博研究所長ならびに内田俊一研究部長に感謝します．

参考文献

[1] A. Goto et al., "Overview of the Parallel Inference Machine Architecture (PIM)", In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, pp.208–229, (1988).

[2] 山本 礼己 他，"並列推論マシン PIM における抽象機械語 KL1-B の実装 – 高級機械語を実装するための道具立 –"，信学技報 CPSY Vol.89 No.168, pp.51–56, (1989).

[3] K. Ueda, "Guarded Horn Clauses: A Parallel Logic Programming Language with the Concept of a Guard", ICOT Techincal Report 208, (1986).

[4] 今井 明 他 "共有メモリ結合マルチプロセッサにおける KL1 向き並列実行 GC 方式の評価"，情処学会研究報告 89-ARC-79-7, pp.49–56, (1989).

[5] A. Sighal and Y. N. Patt, "Unification Parallelism: How much can we exploit?", In *Proceedings of the North American Conference on Logic Programming*, Cleveland, USA, pp.1135–1147, (1989).

[6] M. Furuichi et al., "A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI", In *Proceedings of the Second ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, Seattle, USA, pp.50–59, (1990).

[7] M. Sato and A. Goto, "Evaluation of the KL1 Parallel KL1 Parallel System on a Shared Memory Multiprocessor", In *Proceedings of IFIP Working Conference on Parallel Processing*, Pisa, Italy, pp. 305–318, (1988).

[8] 中川 貴之 他，"プロセス間ソフトウェア割り込み処理を高速化するスリットチェック機構"，情処学会研究報告 89-ARC-77-3, (1989)

[9] J. Crammond, "Implementation of Committed Choice Logic Programming Languages on Shared Memory Multiprocessors", PhD thesis, Heriot-Watt University, Edinburgh, (1988).

[10] 清水 肇 他，"並列推論マシン PIM – 共有メモリ構造クラスタにおけるユニフィケーション"，情処学会第 34 回 (昭和 62 年前期) 全国大会, 2P-3, (1987)