PROLOGの大規模並列処理モデルについて
(ON A LARGE SCALE PARALLEL COMPUTING MODEL OF PROLOG)

安原宏　小松英二
（沖電気工業株式会社　総合システム研究所）

§1 Introduction

論理型言語PROLOGの処理系としては、Warren[0]によって作られたPLMアーキテクチャが逐次型モデルとしてよく知られており、DEC-10上にインプリメントされている。PROLOG言語は並列処理を内在しているため、並列モデルの研究も各地で活発になっている。第五世代コンピュータではPROLOG系の言語が核言語になっており、高速処理の要求が強まるものと予想される。並列処理の研究として筆者の知っている所では表1のものがある。

Table 1 Parallel Computing Model of Prolog

発表者・年	実現手段	並列方式	概要
中島（東大）1980	ソフトウェア	AND	LISPによるACTORモデルでインプリメント.[1]
古川（電総研）1982	ソフトウェア	AND/OR	SIMULAによるAND process, OR process.[2]
相田（東大）1982	ハードウェア	OR	TOPSTARマシン上でインプリメント.[3]
雨宮（武蔵野通研）1982	アーキテクチャ	AND	データフローグラフを用いて共有変数を方向付ける.[4]
梅山（電総研）1982	モデル	OR	コピー, マージ, ユニファイ用の3要素で構成.[5]
後藤（東大）1982	アーキテクチャ	OR	多数のユニファイプロセッサがゴールプールからゴールフレームを取り出す.[6]
田村（神戸大）1982	ハードウェア	OR	ストリーム並列を親子プロセスでインプリメント.[7]
安原（沖電気）1982	ハードウェア	OR	データフローマシン上での実験.[8]
Conery（カリフォルニア大）1981	モデル	AND/OR	ANDプロセス, ORプロセスによるストリーム型モデル.[9]
Wise（ニューサウスウェールズ大）1982	モデル	OR	動的フレームを親子間でパケット転送するモデル.[10]
Shapiro（イスラエル）1982	モデル	AND/OR	変数の入出力モードを利用. PROLOGでインプリメント.[11]

一般に並列処理モデルで問題になる所は、どのようなデータを共有するかである。これは並列度が増大するにつれて重要になって来る。従って小規模な並列処理から大規模になるとき、その処理モデルも変化することを暗示している。アクセス頻度の高いプログラムやデータを共有するモデルでは、10台前後がリミットだと予想される。100台クラスの並列処理では、共有データを避けるモデルを考えなければならないであろう。

小規模並列モデルとして、データフロー計算機を想定して1つのモデルを考える。そこではPROLOGのスケルトンデータは構造体用の共有メモリに置える。大規模並列モデルでは、共有データを一切排除し、完全にローカル処理ができるようなモデルを考える。両者ともOR並列で実行される。

§2 Execution of a PROLOG program on a data flow machine D³P

D³P（Distributed Data Driven Processor）は、低レベルデータフロー処理装置で現在4台のPE（Processing Element）で動作中である.[12] 図1にそのブロック図を示す。命令形式は2入力2出力で、命令レパートリとしては、算術論理演算, 比較, 関数コール, コピー・ゲート・スイッチ等のデータフロー特有の命令, 構造体

メモリへのRead/WriteとAlloc/Free命令等からなっている。その特徴は、トークンに最大1K近色が付くことであり、色を動的に確保/解放することにより論理的に1Kレベルの並行処理が可能なことである。

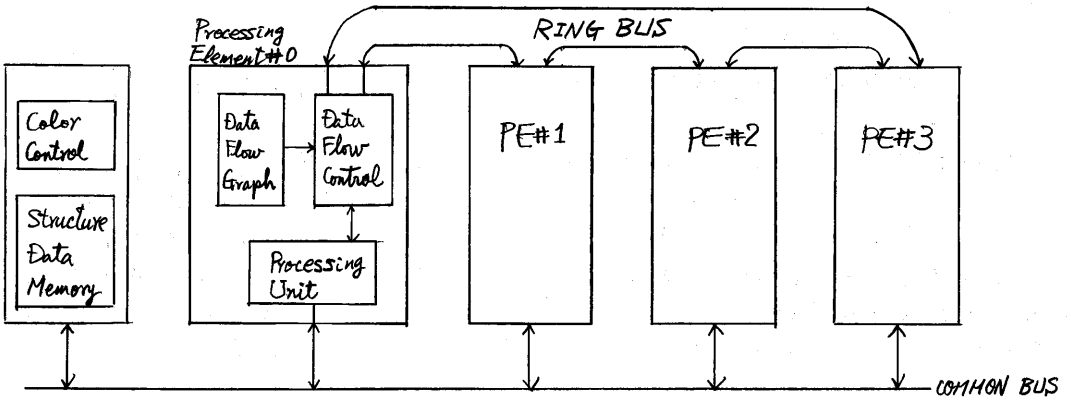


Fig. 1 Configuration of D³P

D³PでのPROLOGの実験は、[9]データフローグラフにPROLOGをコンパイルして動かすことを原理的に確かめることを第1としたため、非常に簡単な例題を用いた。(図2) つまりアーギュメントの個数は1個でかつタイプとしては1文字定数と変数のみを用いた。ただし、PROLOGの特徴であるバックトラックを発生させるクローズにしてある。[6] 図で実線矢印は静的アーク, 破線矢印は動的に出現するアークである。

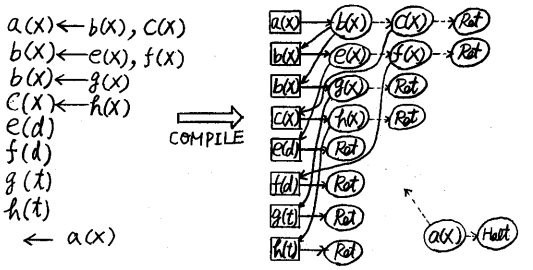


Fig. 2 Prolog Source and Compiled Macro DF Graph

プログラムはインタプリートされるのではなく、直接DFグラフにコンパイルされ実行される。コンパイルは定型的処理(マクロ・テンプレート)で可能である。その種類は、ヘッド, ゴール, リタンで十分であり、マクロへの入出力個数は、アーギュメントの個数により増減する。ヘッド・テンプレートの内部は、アーギュメントタイプにより異なるユニフィケーションコードになる。図3にそのグラフ内部の例を示す。

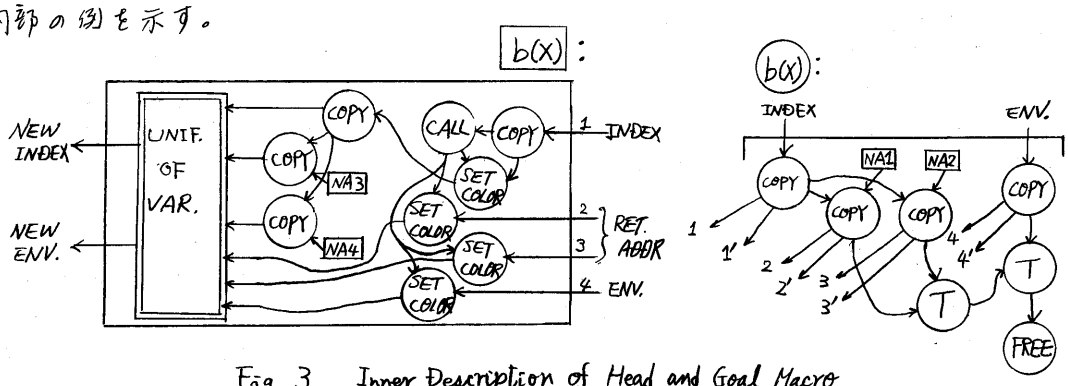


Fig. 3 Inner Description of Head and Goal Macro

ユニフィケーションの入力は、INDEX(これは親ゴールのヒストリをリンクするもので具体的には図1のSDMのアドレスである.)，ゴール側のアーギュメント，及び成功した場合の戻り番地が入る。今回の実験では変数と1文字定数しか出現しないため、変数は-1(Undef)で，定数はASCIIコードでトークン上の値とした。変数のユニフィケーションはゴールアーギュメントを無条件に出力し、定数のユニフィケーションでは一致チェックを行ない成功した場合のみUVAR部に渡す。UVAR部では、INDEXと2つのRET. ADDR. の保存先をAlloc命令で確保し、ストアする。一般的にヘッドの出力は新しいINDEXとENV.である。

図2,3から類推されるように、本実験はOR並列で動作する．更に注目すべきことは，データフローの色つきトークンの概念が十分生かされていることである。つまり複数アーギュメント間で変数の参照がなければ並行に動作可能なことである。色を効率よく管理するため、INDEXを導入したことも重要な点である。それによりユニフィケーション毎に色を獲得/解放が可能になった。

§3 A small scale parallel computing model of PROLOG (SSPCM)

上記実験は非常に限定したプログラムであったが、これを拡張することにより小規模な並列度をもったPROLOGの計算モデルがデータフローマシンを使って得られる。但しここでのデータフローマシンはDFPそのものでなく、データフロー原理に基づいて動作するマシンを考える。以下の説明の中でマシンの特徴についても言及する。

具体的説明に入る前に、SSPCMの原則として設定したものを掲げる。§2の実

① OR Parallel
② Streamless
③ Structure Sharing
④ Variable-Token Driven

験をP(X,Y,Z):-Q(X),R(Y),S(X,Z)の如く2個以上のアーギュメントに拡張する場合を考よう。Xはユニフィケーションの結果を直接Q(X)に渡すことができるが，Yは直接R(Y)に渡せない。何故ならYを渡すことは現在の色をもったトークンとしてR(Y)に入力することであり、Q(X)に伴う一連のマッチングが成功した後、R(Y)に起動がかかるのであるが、そのためには，色を一致させる必要上、色を保持しておかねばならない。これは色の有効利用に反し、②の原則にも反する。色の獲得/解放を1つのユニフィケーション毎に行うことは、本モデルの特徴の1つであり、DFグラフでゴールをスキップするトークンの渡し方は禁止するようにしている。従ってYはSDMにストアしておく必要がある。これはR(Y)のエントリにおいて、SDMのRead Nodeをつけることに対応させる。この動作を一般的に述べると、まずユニフィケーションの結果－Q(X)には新しいINDEX，X,Y,Zとスルーの親フレームポインタが入力トークンとなる。Q内では自クローズ用の変数領域を確保し、Y,Zはそこにストアする。Qが成功すると－R(Y)へは、INDEX,X及び自クローズ変数領域の先頭アドレス(CF)が入力トークンとなる。このときはCFに従って現変数を読み取ると共に新たなCFをSDM上に確保し、Xはそこにストアする。Zは読み取った値を新たなCFにストアする。Yの値は読みとった値を使用する。2番目以降のゴール項はRと同様に読み取りと確保を行なう。毎回新たに変数領域を確保することにより、複数のストリームが

独立に動作可能となる。（原則②）図4はこのクローズをDFグラフで概念的に表示したものである。

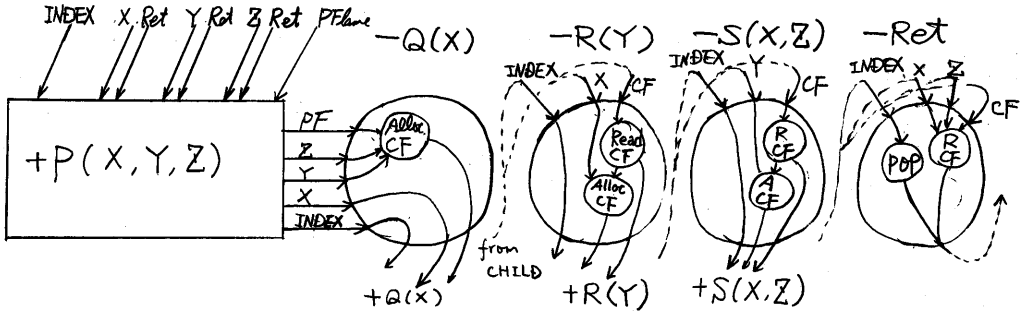


Fig.4 General Description by Data Flow Graph of P(X,Y,Z):-Q(X),R(Y),S(X,Z)

このクローズのRetノードではINDEX,X,Z,CFがトークンとして入ってくるが、CFアドレスに従ってYを読みだしてからINDEX先のreturn情報部に保持しているX,Y,Zの戻り先に従ってbindされたX,Y,Zを親へ返す。このときヘッドで確保した領域を解放する。以上により変数がローカルのみの場合への拡張は、ここまでのSDM情報に変数領域を付加することにより成就する。

次にスケルトンが含まれるクローズへの拡張を考える。この場合には大きな問題が発生する。図4でうまくいった理由は、ローカル変数のみのときには、クローズの親子間のリンクが1対1に対応しているので親から子，子から親への移行にも環境の受渡しがスムースにできた。スケルトンの場合はそれができなくなる。図5はその様子を示している。Value Tokenの場合はコピーによってOP1,OP2に入力するが、それぞれ独立に処理できる。ところがSkeleton Tokenの場合はコピー命令で分岐しても元のデータは共用メモリに存在しているため、OP3とOP4が書き込み動作をすると干渉が発生する。スケルトンの入ったプログラムでは、フレーム間のリンクが必要なため図4のようにAllocを1回だすだけではすまなくて、それに結びついているすべてのフレームもコピーしなければならない。これを自動化するには、構造体メモリ内にも色の概念を導入する必要があり、データフローアーキテクチャの検討課題となる。

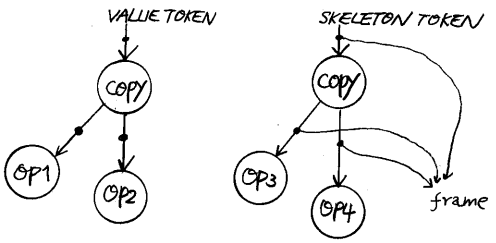


Fig.5 Difference of Value Token and Skeleton one.

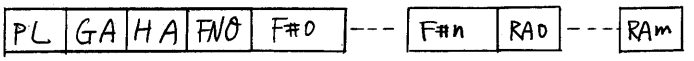
SSCPMの原則④は変数がDFグラフ上を値を持ちながら流れることで、DFとPrologの親和性を示す。Prologの変数がデータのキャリアにすぎないことと、DFグラフのアークがトークン・パスとして機能していることはその著しい類似点である。このため参照トークンが不用であり、アークで結ばれているものと解釈すればよい。従ってトークンのタイプとしては、数値，アトム，分子となる。分子はスケルトンへのポインタ（SDMあるいはローカルメモリのいずれでもよい。）とフレームへのポインタ（SDM）からなっている。

§4 A large scale parallel computing method of PROLOG (LSPCM)

大規模な並列処理にはPROLOGに限らず共通的課題がある。1つは通信オーバヘッドであり、もう1つはローカリティをいかにして獲得するかである。本論では具体的並列処理アーキテクチャを示さないので前者に関しては仮定せざるを得ないが、システム像としては、クロスバSW等でスター状に結合されたマルチプロセッサを想定すればよいと考えている。後者に関しては以下のモデルで具体的に示す。

PROLOGと並列処理の親和性に関しても、従来の言語にみられない良さがある。これまで並列処理は主にアーキテクチャ面から研究され、その後ソフトウェアで如何に使いこなすかという展開が多かった。データフローの計算モデルはもう少しソフトウェア的観点に立ってできたが、言語モデルとしては未だプリミティブであった。一方PROLOGはロジック用プログラミング言語として出現して来たのであるが、従来のプログラミング言語と異なり、順序制御という概念をもっていないので、その計算モデルは自由度の大きいものとみなせる。つまり逐次制御を強制しないので並列処理が容易であるともいえる。この言語の可能性を見通すこと(時期尚早であろうが、パターンマッチングによる推論機能は一歩人間に近づけるものであり、そういう機能を並列処理により効率よく実行することにより本来の威力をだすものであろう。このことは第5世代コンピュータプロジェクトで、PROLOGと並列処理アーキテクチャが中心課題となっていることでもうかがわれる。

多くのSSPCMは共有メモリ上にフレームを作ったが、LSPCMではそれを避けるようにする。まず処理系のモデルとして多数の処理装置がネットワークを介して相互に結合しているものとする。各処理装置は入力キューを持ち、プログラムはローカルメモリ上に全ての処理装置とも同一の像を有する。データフローモデルとの相違点は共有の構造体メモリをもたないことである。言語の処理モデルの原則としてはSSPCMの①②③は同じであるが、変数駆動ではなくて④フレームトークン駆動を設定する。フレームトークンとはパターンマッチングに必要な全情報をパケットとして送出することを意味する。図6は本モデルのシミュレーションに使用したパケット形式である。OOR並列の原則により、パターンマッチングの対象となる全クローズにパケットを送出する。送出先のアドレスは1つは自分自身に送り、他は乱数を発生して決定することにした。これにより負荷が平等に分布するものと予想される。②のStreamlessの考え方はSと同様である。パケット方式により、INDEXの概念は不用になる。③のストラクチャシェアリングの考え方は、パケット長の極小化、パケット組立て・分解時の効率性から採用した。各処理装置内ではパケット用のメモリを動的に確保して、送出後は解放することができるため常時GCを行なっているものとみなせる。



PL: Packet Length
GA: Goal Address
HA: Head Address
FNO: Frame Number
F#i: i-th Frame
RAj: j-th Return Address

Fig.6 LSPCM Packet Format

§5 Simulation of the LSPCM

本モデルの妥当性を評価するため計算機上でシミュレータを作成中である。このモデルのコードは Code Control Field (CCF) と Code Value Field (CVF) からなりCCFはビット毎に意味を持たせ、ヘッダ，ゴール，アーギュメントエンド，アトム，整数，L/G変数，スケルトン，Return Flag 等からなっている。CVF部はCCF部によって意味が異なり、ヘッダでは当該クローズの変数個数を示し，ゴールでは対応するヘッドアドレスの集合になっている。このコード系は Warren[6] のPLMコードに類似しており、固定長でソースプログラムに対応するシンプルなものである。コンパイラはまだ作っていないのでハンドコンパイルしたものをシミュレータ中に組み入れた。プロセッサ数は10台（これな任意にセットできる.）とし、ラウンドロビンでスケジューリングを行なった。図7はシミュレータの構成を示している。

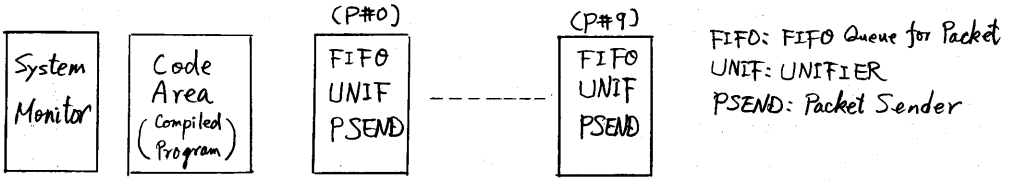


Fig.7 Simulator of LSPCM

パケットはパケット長のメモリを動的に確保して、相手先FIFOにその先頭アドレスをセットすることで代用している。パケットメモリはシステムプログラムをコールしてとるため図には示していない。各プロセッサ内での処理はFIFOから1つのパケットをフェッチし、分解する。パケットとコードエリアによってパターンマッチングを行なった後成功すればOR並列の数だけ乱数を発生させ、その値により送出先プロセッサを決定する。ユニファイアでは、新たに自クローズ用のフレームを生成し、初期化してからマッチングに入る。フレームの形式はコード形式と同様に Frame Control Field (FCF) と Frame Value Field (FVF) からなり．FCFはフレーム開始／終了，アトム・整数・スケルトンのタイプ情報等からなっている．FVFはFCFに対応して意味を持っており、フレーム開始では、自フレームの番号，全変数個数，ローカル変数個数を持っている。タイプ情報に対する値はPLMと同様の表現形式である。たとえばスケルトンではフレームアドレスとスケルトンポインタからなる。パターンマッチングの成功後、古いパケットと自フレームを1つのパケットに組立てるが、次のゴールが Ret 命令であれば，グローバル変数に相当する所だけ残して他は捨てる。そのためフレームの構成は前半にグローバル変数を対応させるようにアドレッシングしてある。複数の Ret 命令が連続するときは、その回数だけローカルフレームの削除作業が入る。

本シミュレータによる評価情報の収集は今回の発表ではできなかったので後日その結果を報告する予定であるが、シミュレータのデバッグをかねて、Appendのプログラム： AP(⌴, X, X) ←; AP(A|X, Y, A|Z) ← AP(X, Y, Z) ; を実行文： ?AP(a|b, c|d, Z); で評価した。パターンマッチングは (P#0, P#1), (P#1, P#9), (P#9, P#6) の順序でOR並列が進行した。パケット長はそれぞれ8語，15語，30語になった。机上の計算によると $(a_1, \cdots, a_n)$ のリバースプログラム：

R(x|L0,L)←R(L0,L1),AP(L1,(x),L); R(␣,␣)←の場合、全てのフレームは n(リバース分)$+\frac{n(n+1)}{2}$(Ap分)段のオーダになり非常に増大する危険があるが、この例の場合はモード指定を導入するとリードオンリのグローバル変数はRet時に残しておく必要がなくフレームは減少する。

§6 CUT operation in the parallel computing model

OR並列をカット演算子を含むプログラムに適用すると、矛盾を生じることがある。例えばCASE文的な用例：A(a)←!…; A(b)←!…; A(c)←!…; A(x)←P(エラー); の場合がそうである。一方 $P_1 \leftarrow Q_1!\cdots$; $P_2 \leftarrow Q_2!\cdots$; $P_3 \leftarrow Q_3!\cdots$; のような場合 Q_1, Q_2, Q_3 が離散的な解集合をもつならば、OR並列で実行しても無駄があることを除けば問題ではない。このような場合は並列に処理する以上避けられないこととともいえる。

Wise[10]は条件AND,条件ORを導入してOR並列のカットを実現しているが、本稿で述べたOR並列は1つのゴールの成功/失敗が明示的でないことを特徴としている。この利点はリソースが有効に使用できることであり、失敗したプロセスはその時点で消滅する方法をとっている。従ってこのOR並列では、カットの左側ゴールが1回のマッチングで成功/失敗が判明する場合のみ対応できる。例えば P←!Q,R,…型の場合や、P←Q,R,S!…でQ, R, Sの結果が1回のマッチングで判明する場合(Q,R,Sがそれぞれ、組込み述語かアサーション)である。一般の場合にどのような手段を提供するかは検討中である。

§7 Conclusion

本論では小規模、大規模並列に分けて2通りのOR並列モデルを示した。共通点としてストリーム並レの方式をとり、できるだけプロセッサが自由になるようにしていることと、ストラクチャシェアリングをとったことである。相違点は共有メモリの有無で名前で1、全てパケットにして転送している。パケット長の増大は通信オーバヘットをひき起すが、それに関する評価は現在シミュレータで採取中である。特に興味ある所は、並次型、小規模、大規模モデルの性能比較であるがここでは論じてない。大規模並列アーキテクチャに関しては、提示していないが、条件として相互独立なネットワークであり、パケット長が数十語から数K語の可変長であることが挙げられる。第五世代コンピュータプロジェクトでは、並列推論マシンが研究テーマの1つになっており、並列Prologの研究と相俟って、80年代において実用的な並列マシン(〝スーパロジックマシン〟)が誕生することが期待できよう。

[謝辞]

本研究の機会を与えて下さった総合システム研究所安楽所長、似鳥部長、研究推進に協力して下さった河村室長、来往研究員をはじめとするデータフロー計算機の研究グループ員、有益な意見交換をしていただいたICOTの伊藤研究員にそれぞれ感謝いたします。

[参考文献]

(0) D. Warren: Implementing PROLOG, D.A.I Research Report NO. 39
Department of Artificial Intelligence University of Edenburgh, 1977, May

(1) H. Nakashima: Parallel Prolog,
東京大学情報工学・修士論文, 1980

(2) K. Furukawa, K. Nitta, Y. Matsumoto: Prolog Interpreter based on Concurrent Programming, Prolog Conference, 1982

(3) 相田仁: 並列Prologシステム"Paralog"について
Prolog Conference, 1982

(4) 雨宮真人, 長谷川隆三, 橋爪正樹: データフロー概念に基づく推論実行方式の研究, 電子通信学会電子計算機研究会, EC82-30, 1982

(5) 梅山伸二: 論理プログラムの並列実行について,
情報処理学会人工知能と対話技法, 26-5, 1982, 6月

(6) 後藤厚宏, 相田仁, 田中英彦, 元岡達: 推論向き高並列計算機システムの基本アーキテクチャ, 電子計算機研究会, EC82-43, 1982

(7) 田村直之, 有尾隆一, 松田秀雄, 金田悠紀夫, 前川複男: K-Prolog: 並列マシンエでのPrologの実現, 記号処理研究会, 20, 1982, 10月

(8) 安原宏, 来住晶介, 河村保輔, 伊藤貴義: データフロー計算機DDPとPROLOG言語, 情報処理学会第25回全国大会, 1982

(9) J.S. Conery, D.J. Kibler: Parallel Interpretation of Logic Programs, Proc. of the 1981 Conference on Functional Programming Language and Computer Architecture, ACM.

(10) M.J. Wise: A PARALLEL PROLOG: the construction of a data driven model, 1982 ACM Symposium on Lisp and Functional Programming, 1982, Aug.

(11) Shapiro: In Concurrent Prolog,
ICOT講演資料, 1982, 11月

(12) 伊藤貴義, 来住晶介, 安原宏, 河村保輔: データフロー計算機DDPの実験システム, 情報処理学会アーキテクチャ研究会, 1981, 7月