ＰｒｏｌｏｇマシンＰＥＫ上でのｐｕｒｅ　Ｐｒｏｌｏｇインタプリタの作成

田村直之*、和田耕一*、松田秀雄*、金田悠紀夫**、前川禎男**

*　神戸大学大学院自然科学研究科　　**　神戸大学工学部システム工学科

1　はじめに

本稿では、逐次実行型Prologマシンｐｅｋ上に開発したPrologインタプリタ第1版の構成と、テストプログラムの実行結果について報告する。

テストプログラムには、2つのリストを連結するプログラムとリストの逆転を行うプログラムを使用した。これらのプログラムの実行結果によれば、本インタプリタは、DEC 2060上のDEC-10 Prolog コンパイラとほぼ同等の、約40K LIPSの性能を持つ。

2　PrologマシンＰＥＫの特徴

逐次実行型PrologマシンＰＥＫは、Prologのプログラムを高速に実行するためのマシンアーキテクチャの研究を目的とした実験機であり、

- シーケンシャル実行
- マイクロプログラム制御
- ストラクチャシェアリング

などを、基本設計方針とし、さらに

- タグアーキテクチャ
- メモリの分散化と低レベル並列処理
- マッチング回路
- 自動トレイル回路
- 自動アンドゥ回路

などの、Prolog専用のハードウェア機能を備えている[1～4]。

文献[1～3]以降に、いくつかのハードウェアについて改良を行った。

① UNDEF フラグの追加。以前のハードウェアでは変数値が未定義であることを示すUNDEF フラグが一つしかなく、レジスタFT1 で指される変数セル、あるいはFT2 のどちらかの状態しか示せなかった。そこで、UNDEF フラグを追加して、UNDEF1,UNDEF2 の2つにした。フラグは、マイクロ命令中のMUX フィールドで指定することで、自動的に設定される。

② フレームレジスタの追加。構造体の要素の読み込み時に、グローバル変数のフレームアドレスをフレーム部に自動的に付け加えるための、フレームレジスタを追加した。共有メモリへのアドレスレジスタAD1,AD2 に構造体の先頭アドレスを書き込む時に、フレーム部の値をグローバル変数のフレームアドレスにしておくと、データレジスタRD1,RD2 から読み込む時に、設定したグローバル変数のフレームアドレスがフレーム部に自動的に付け加えられる。

③ 共有メモリの疑似2ポート化。共有メモリには2つのアドレスレジスタ、2つのデータレジスタが用意されているが、以前のハードウェアではアクセスをオーバーラップできなかった。そこで、共有メモリを疑似的に2ポート化することにより、アドレスの設定、データの読み込み、データの書き込みを2つのポートでオーバーラップできるようにした。

④ イミディエイトジャンプ機能の追加。イミディエイト値によるジャンプを、Sバスを使用せずに行えるようにした。

3　Prologの言語仕様

各ハードウェア機能の性能を調べるために、ＰＥＫ上にPrologインタプリタ第1版を開発した。Prologインタプリタ第1版では、pure Prolog にcut の機能が付け加わった言語を対象とし、さらに

- 変数はすべてグローバル変数とする
- indexing[5], tail recursion optimization[6] などの最適化は行わない

ことにした。ただし、ローカル変数のフレームの管理は

行っている。

また、上記のハードウェア改良点のうち、③と④は利用していない。

4 Prologインタプリタ

Prologインタプリタ第１版では、以下のようにメモリを利用した。

- ＷＣＳ
  Prologインタプリタの格納。
- 共有メモリ
  Prologプログラム、アトムヘッダなどの格納。
- プロセスメモリ
  コントロールフレームの格納。
- グローバルスタック
  グローバル変数、ローカル変数の格納。
- トレイリングスタック
  アンドゥ作業用のトレイル情報の格納。
- ハードウェアスタック
  ユニフィケーションで使用。
- レジスタファイル
  大域的な情報の保存などに使用。

インタプリタは各述語の実行ごとに再帰的に呼び出され、必要なら実行の管理情報を保存するためのコントロールフレームを作成する。インタプリタの呼び出しには以下のようなパラメータを使用する。

- 成功時戻り点(success return point)：述語の実行が成功した場合に制御を戻すマイクロアドレスとコントロールフレームのベースアドレスを示す。
- 失敗時戻り点(failure return point)：述語の実行が失敗した場合に制御を戻すマイクロアドレスとコントロールフレームのベースアドレスを示す。
- 呼び出し側グローバルフレーム(caller global frame)：呼び出し側のグローバル変数のフレームアドレスを示す。
- 呼び出し側ローカルフレーム(caller local frame)：呼び出し側のローカル変数のフレームアドレスを示す。
- 述語名(predicate name)：インタプリタが実行する述語名を示す。
- 引数リスト(argument list)：実行する述語の引数のリストを示す。

成功時戻り点と失敗時戻り点は、復帰先のマイクロアドレスとコントロールフレームアドレスの２語からなっており、それぞれDEC-10 Prolog での、continuationとlastest choice pointに相当する。復帰先の指定が、マイクロプログラムレベルで行えるため、マイクロコードにコンパイルした述語などからインタプリタを呼び出すことが容易になる。また、インタプリタは述語実行が成功した場合には、リターン情報として

- 最終選択点(latest choice point)

をレジスタに設定する。最終選択点は、バックトラックの際に実行を再開する点を示すもので、実行を再開する場所のマイクロアドレスとコントロールフレームアドレスの２語からなっている。したがって、インタプリタは実行している述語の選択(alternative) が残っている場合は、最終選択点のコントロールフレームを自分自身のコントロールフレームと同一にし、選択が残っていない場合は、最終選択点を失敗時戻り点と同一にする。

述語名としては、リテラルアトム（タグはlit ）か、組み込み述語プログラムの先頭マイクロアドレス（タグはcode）が可能である。リテラルアトムの場合は、アトムヘッダ内に述語定義へのポインタが書き込まれている。述語定義としては、ユーザ定義述語へのポインタ（タグはclause）か組み込み述語プログラムの先頭マイクロアドレス（タグはcode）が可能である。なお、cut は組込み述語として処理されている。

コントロールフレームには、以上のパラメータのほかに

- 実行中のclauseへのポインタ
- alternative のclauseへのポインタ
- 使用しているグローバルフレームのアドレス
- 使用しているローカルフレームのアドレス
- 述語の実行開始時のトレイルスタックポインタ
- 実行中のclauseのグローバル変数の個数

・ 実行中のclauseのローカル変数の個数

が格納されている（計15語）。

4 テストプログラムと実行結果

テストプログラムとしては、長さｎのリストを空リストに連結するappendプログラムと、長さｎのリストを逆転させるreverse プログラムの２つを実行した。

・ appendプログラム

```
append([],Z,Z).
append([W|X],Y,[W|Z]) :- append(X,Y,Z).
?- append([1,2,3,..,n],[],Z).
```

・ reverse プログラム

```
reverse([P|Q],R) :-
        !, reverse(Q,S), append(S,[P],R).
reverse([],[]).
append([W|X],Y,[W|Z]) :- !, append(X,Y,Z).
append([],Z,Z).
?- reverse([1,2,3,..,n],R).
```

表１にそれぞれのゴール文を実行した時の、実行命令数を示す。また表２には、特にｎ＝30の場合の実行命令数と、実行速度を示す（ただし、１マイクロ命令の実行速度を平均 170ナノ秒としている）。また、appendプログラムとreverse プログラムの処理単位ごとの命令数を表３と表４に示す。ここで、それぞれの処理単位名は

```
init  : goal実行前の初期化
pred  : 述語のタイプ判断
ent   : enter 処理
try   : try 処理
unif  : unification 処理
tryl  : try last処理
neck  : neck処理
cut   : cut の呼び出しと実行
call  : 述語の呼び出し
calll : 再右端の述語の呼び出し
foot  : foot処理
ret   : goal実行の終了
```

を意味する。

表１と表２から、どちらのプログラムについても、一回の推論当たり 145マイクロ命令で実行でき、速度も約 40K LIPS(Logical Inference Per Second)であることが分かる。これは、現在実用的な処理系としては最も速いとされているDEC 2060上のDEC-10 Prolog コンパイラとほぼ同等の速度である。

これは、ＰＥＫと同様の回路技術を用いているLISPマシン上のPrologインタプリタが、10～20K LIPSであるこ

表１． append, reverse での実行命令数

	推論の回数	実行した命令の総数	コントロール関係の命令数	ユニフィケーション関係の命令数
append([1,2,...,n],[],Z)	$n + 1$	$145n + 149$	$49n + 94$	$96n + 55$
reverse([1,2,...,n],R)	$\frac{1}{2}n^2 + \frac{3}{2}n + 1$	$\frac{145}{2}n^2 + \frac{357}{2}n + 178$	$33n^2 + 85n + 101$	$\frac{79}{2}n^2 + \frac{187}{2}n + 77$

表２． append, reverse での実行命令数（n=30、一命令＝170nsec.）

	推論の回数	実行した命令の総数	コントロール関係の命令数	ユニフィケーション関係の命令数	一秒当たりの推論の回数
append([1,2,..,30],[],Z)	31	4499	1564	2935	40.5K LIPS
reverse([1,2,..,30],R)	496	70783	32351	38432	41.2K LIPS

表3． appendプログラムの実行命令数の内訳

	init	pred	ent	try	unif	tryl	unif	neck	calll	foot	ret
g(n)	12	5	14			5	13	8	8+a(n)	4	1
a(0)		5	14	5	44			8		5	
a(n)		5	14	5	21	5	65	8	8+a'(n-1)	4	
a'(0)		5	14	5	52			8	8+a'(0)	5	
a'(n)		5	14	5	24	5	72	8	8+a'(n-1)	4	

g(n) ： goalを実行した時の総命令数
a(n) ： append([1,2,...,n],[],Z)を実行した時の総命令数
a'(n) ： append(X,Y,Z) を実行した時の総命令数

表4． reverse プログラムの実行命令数の内訳

	init	pred	ent	try	unif	tryl	unif	neck	cut	call	calll	foot	ret
g(n)	12	5	14			5	13	8			8+r(n)	4	1
r(0)		5	14	5	21	5	36	8				7	
r(n)		5	14	5	58			8	7+12	8+r'(n-1)	9+a(n-1)	6	
r'(0)		5	14	5	24	5	39	8				7	
r'(n)		5	14	5	61			8	7+12	8+r'(n-1)	9+a(n-1)	6	
a(0)		5	14	5	24	5	52	8				7	
a(1)		5	14	5	74			8	7+12		9+a'(0)	6	
a(n)		5	14	5	78			8	7+12		9+a'(n-1)	6	
a'(0)		5	14	5	24	5	53	8				7	
a'(1)		5	14	5	75			8	7+12		9+a'(0)	6	
a'(n)		5	14	5	79			8	7+12		9+a'(n-1)	6	

g(n) ： goalを実行した時の総命令数
r(n) ： reverse([1,2,...,n],Z)を実行した時の総命令数
r'(n) ： reverse(Q,S)を実行した時の総命令数
a(n) ： append(S,[P],R) を実行した時の総命令数
a'(n) ： append(X,Y,Z) を実行した時の総命令数

とを考えると[7~9]、ＰＥＫのアーキテクチャがPrologプログラムの実行速度を２～４倍程度改善したと言える。

５　ハードウェアの評価

５．１　仮想マシン

Prologプログラムの実行における、各ハードウェアの貢献度を調べるために、ＰＥＫからProlog用のハードウェア機能を取りのぞいた仮想マシン（以下ではＶＭと呼ぶ）を設定し、その上でのappendプログラムの実行過程とＰＥＫでの実行過程の比較を行った。

ＶＭのアーキテクチャとしては、

- 垂直型マイクロ命令。マイクロ命令には分岐命令と演算命令があり、演算命令はＰＥＫと同様の２レジスタ演算である。
- データ幅はタグ部 4ビット、バリュー部16ビットの計20ビットでフレーム部はない。
- タグによるディスパッチ機構。ディスパッチは多方向分岐命令を含めて２命令で行えるが、２つのタグによるディスパッチ機構は備えていない。
- 専用のハードウェアスタックを１つ備えている。スタックポインタ設定後、スタックトップのリード／ライト、プッシュ／ポップが可能である。
- 主メモリに対しては、アドレスレジスタ設定後、リード／ライトが可能になる。ただしリード／ライトは２サイクル必要とする。

を想定した。さらにインタプリタの構造に関しては、

- ＰＥＫと同一のデータ構造をとる。
- 全体的な処理の流れは、ＰＥＫのインタプリタと同じにするが、局所的な部分は最適化する。
- ＰＥＫのプロセスメモリとハードウェアスタックをＶＭのハードウェアスタックに割り当て、他のスタック類は主メモリを利用する。

などを仮定した。

５．２　仮想マシンでの実行結果

上記のような仮定のもとで、appendプログラムのループの部分（表３でのa'(n)）をＶＭ上でトレースしたところ 300命令となった（ＰＥＫでは 145命令）。ＰＥＫとＶＭの比較結果を表５に示す。ただし、ＶＭ側のプログラムでは、無条件分岐などのむだな命令は除去し、主メモリのリード／ライトは２命令として数えている。

ＶＭの１命令を 170ナノ秒とすると、約19.6K LIPSであり、LISPマシンでの結果に比較的良く一致する。

表５．　ＰＥＫとＶＭの命令数の比較

処理名	ＰＥＫ	ＶＭ	比率
pred	５	６	1.20
ent	１４	１５	1.07
try	５	８	1.60
unif	２４	３９	1.63
tryl	５	９	1.80
unif	７２	１８４	2.56
neck	８	１４	1.75
call	８	１７	2.13
foot	４	６	1.50
合計	１４５	３００	2.07

表６．　ＰＥＫで減少した命令の内訳

ハードウェア機能	減少命令数	割合	順位
水平型マイクロ命令	２４	15.5	③
34ビット転送	１０	6.5	⑤
プロセスメモリ	６	3.9	⑦
パイプライン	５５	35.4	①
フレームレジスタ	６	3.9	⑦
マッチング回路	９	5.8	⑥
グローバルスタック	３０	19.3	②
自動トレイル回路	１６	10.3	④
自動アンドゥ回路	－１	-0.6	⑨
合計	１５５	100.0	

さらに、比較結果からみると、VMでのunification処理はＰＥＫの約2.3 倍かかっており、ＰＥＫのアーキテクチャがunification の高速化に大きく貢献したと言える。

５．３　各ハードウェア機能の評価

ＰＥＫの命令数とＶＭの命令数の差は

- 水平型マイクロ命令
- 34ビット転送
- プロセスメモリのアドレス形式
- 構造体のパイプラインリード
- フレームレジスタ
- マッチング回路
- グローバルスタックの専用化
- 自動トレイル回路
- 自動アンドゥ回路

などが要因となっている。ＰＥＫ側で減少した命令を、これらの要因別にまとめると、表６のようになる。

水平型マイクロ命令の採用により、減少した命令数は24命令で、ほとんどがＡＬＵ演算と条件分岐を同時に行っている命令である。

34ビット転送によるものは10命令であり、その内訳は、X=[m,..,n]のdereference で一命令、Z := [W|Z]とX := [m+1,..,n] の代入で二命令、[m,..,n]と[W|Z] のpushとpop で四命令、そのほかEOS 処理で三命令となっている。

プロセスメモリに専用のベースレジスタを設け、ベースレジスタ＋オフセット（+0～+255）のアドレス形式を使用可能にしたことにより 6命令減少している。これは、ＶＭ側では連続したpush/popでアクセスできない場合、スタックポインタの再設定が必要になるためである。

共有メモリからの読み出しのパイプライン化により、減少した命令数は55命令となっている。これは、ＶＭ側では常にアドレスのセットが必要なこと（19命令増加）、次のデータがEOS かどうか判断するための先読み処理を行っていること（14命令増加）、EOS フラグの判断（13命令増加）などが原因である。

フレームレジスタの利用による減少は 6命令となっている。これは、ＶＭのプログラムで変数フレームアドレスの退避（caller側とsource側）を三回行っているためである。

マッチング回路による減少は 9命令となっている。ＰＥＫでは、二つのレジスタFT1 とFT2 に格納されている二つのタグの値とバリュー部の比較結果を使用して16通りの多方向分岐を行う。したがって、単一のタグを使用したディスパッチ機構を二回行う場合に比較して一命令減少し、双方がliteral atomなどの場合は、さらに二命令減少する。ただし、ＰＥＫでは一方のタグのみでディスパッチする機能を省いたため、その場合には反対側に特別のタグを持つタームを書き込む必要があり、逆に一命令増加する。

グローバルスタックの専用化により、減少した命令数は30命令となっている。そのうち、アドレスの設定が10命令、リード／ライト待ちが14命令、UNDEF フラグによるものが 6命令である。ＰＥＫでは、変数の値がUNDEFかどうかは、実際にグローバルスタックから読み出さなくても、フラグで判断できる。

自動トレイル回路による命令数の減少は16命令となっている。これは、ＰＥＫではトレイル処理はマイクロ命令中の１ビットをセットするだけで良いが、ＶＭでは四回のトレイル作業ごとに、主メモリのアドレスセットとライト、トレイルポインタのインクリメントを行っていることによる。

自動アンドゥ回路では、逆に１命令増加している。これは、実行したプログラムが決定的であるため、アンドゥの必要がないこと、また自動アンドゥ作業の終了を調べるための命令が増加しているためである。

以上をまとめると、構造体の要素のパイプライン読み出し機能や、グローバルタック、プロセスメモリ、自動トレイル回路など、メモリやスタックの分散化による効果が大きい（合計で全体の69%）。また、ＶＭではアドレスレジスタへの書き込みが44命令あるが、ＰＥＫでは

共有メモリのアドレスレジスタAD1,AD2 への書き込みが11命令、プロセスメモリのベースレジスタPBR への書き込みが 2命令であり、分散化させたメモリごとに特別のアドレス形式を設けたことが性能向上に貢献したと考えれる。

6　おわりに

逐次実行型専用PrologマシンＰＥＫ上にpure Prologインタプリタを作成し、性能評価を行った。テストプログラムとして用いたappendプログラムとreverse プログラムのどちらについても、推論一回当たり 145マイクロ命令で実行でき、約40K LIPSの結果を得た。これは、現在実用的な処理系としては最も高速とされているDEC 2060上のDEC-10 Prolog コンパイラとほぼ同等の速度である。

また、各ハードウェアの貢献度を調べるため、仮想マシン上のインタプリタとの比較を行った。その結果によれば、ＰＥＫは仮想マシンの二倍以上の性能であり、特にスタックなどのメモリモジュールの専用化による効果が大きいと考えられる。

参考文献

[1] 田村直之、和田耕一、松田秀雄、小畑正貴、金田悠紀夫、前川禎男：シーケンシャルPROLOGマシンＰＥＫのアーキテクチャとソフトウェアシステム、情報処理学会記号処理研究会資料 25-2、1983年10月

[2] 田村直之、和田耕一、松田秀雄、金田悠紀夫、前川禎男：PrologマシンＰＥＫについて、Proc. of the Logic Programming Conference '84, 8-2, Tokyo, 1984年 3月

[3] Y.Kaneda, N.Tamura, K.Wada, H.Matsuda: Sequential PROLOG Machine PEK Architecture and Software System, Proc. of International Worksh op on High-Level Computer Architecture, 4.1, Los Angeles, May 1984.

[4] N.Tamura,K.Wada,H.Matsuda,Y.Kaneda,S.Maekawa: Sequential Prolog Machine PEK, Proc. of the International Conference on Fifth Generation Computer Systems 1984, pp.542-550, Tokyo, Nov. 1984.

[5] D.H.D.Warren: Implementing PROLOG -- Compiling Predicate Logic Programs Vol.1,2, DAI Research Report No.39,40, Dep. of AI Univ.of Edinburgh, 1977.

[6] D.H.D.Warren:An Improved PROLOG Implementation which Optimises Tail Recursion, Proc. of Logic Programming Workshop, pp.1-11, 1980.

[7] 竹内郁雄、日比野靖、奥乃博、大里延康、渡辺和文：ＥＬＩＳ－ＴＡＯの性能評価、情報処理学会第28回全国大会 3F-2 、1984年 3月

[8] 大寺信之、斎藤年史、清原良三、西開地秀和、安井裕：ＥＶＬＩＳマシン上のPrologインタプリタとその動特性、情報処理学会記号処理研究会資料 27-4、1984年 3月

[9] 服部彰、丹羽雅司、篠木剛、木村康則、岸本光弘：PROLOGインタプリタの構成、情報処理学会第29回全国大会 7B-1 、1984年 9月