5U-1

内蔵型高速PROLOGプロセッサIPP (XII)
－ システム統合 －

鈴木弘 (*)　大津善行 (*)　石川隆 (*)　阿部重夫 (日立製作所　日立研究所)
*日立プロセスコンピュータエンジニアリング

1．はじめに

'８０初頭のＡＩブーム以降、ＰＲＯＬＯＧがＡＩの核言語として注目されている。特に、D. ＷａｒｒｅｎによるＰＲＯＬＯＧ命令セットが提案 1) されてからは、実行性能が大きく向上し、ＰＲＯＬＯＧマシンの研究・開発が活性化された。

しかしながら、ＰＲＯＬＯＧの高速実行が実現できた反面、専用マシン化が進み、既存ソフトが流用できないために実用化のスピードが鈍っているのが現状である。

そこで我々は、汎用のスーパミニコンにＰＲＯＬＯＧ高速機構を内蔵し、ハードウェアレベルでの、既存ソフトとＡＩ言語サポートの統合化を実現した（ＩＰＰ）。2)

今回、実機上にＰＲＯＬＯＧシステムを構築したので、その処理方式と性能を報告する。

2．システム構成

図１にＩＰＰ上のＰＲＯＬＯＧシステムの構成を示す。本システムは、高機能なプログラミング／デバッグ環境をサポートするインタプリタ／デバッガ、高速推論をサポートするコンパイラ、及びインタプリタとコンパイラのコード混在実行をサポートするインタフェースにて構成している。

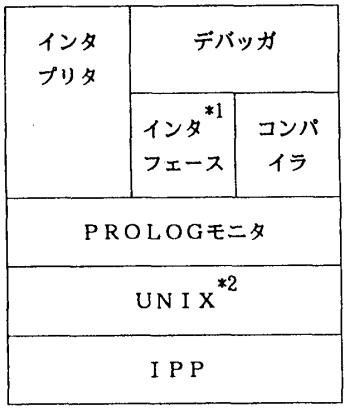

図１　ＰＲＯＬＯＧシステムの構成

*1 インタフェースは、インタプリタコードとコンパイラコードの混在実行をサポートする
*2 ＵＮＩＸは、ＡＴ&Ｔの登録商標である

3．インタフェース処理方式

ＰＲＯＬＯＧシステムをＩＰＰ上にまとめあげる上での重要なポイントであったインタプリタコードとコンパイラコードの混在実行をサポートするインタフェースルーチンの処理方式を示す。

まず、インタフェース処理の概念を図２に示し、（１）～（４）に基本処理方式を示すが、以降インタプリタコードをＩコード、コンパイラコードをＣコードと略して述べる。

（１）ＩコードからＣコードへの実行遷移

ｉｎｃｏｒｅ方式を採用した。これは、述語ａのｐｕｂｌｉｃ宣言により、ａ：－ｉｎｃｏｒｅ（ａ）をＩコードに生成するもので、組み込み述語ｉｎｃｏｒｅは、ａのＣコードへ絶対アドレスジャンプをする。このとき、Ｉコードへの戻りのために、成功時と失敗時に使用するインタプリタローカルスタックのアドレスをインタフ ェーススタックに記憶する。

（２）ＣコードからＩコードへの実行遷移

コンパイル時に未定義であった場合に、常にインタフェースルーチンを呼出すこととした。インタフェースは、該当する述語がＩコードに存在するか否かを見て、あればインタプリタにジャンプし、なければＦＡＩＬする。このとき、（１）同様に、成功時と失敗時のＣコードへの戻りアドレスをインタフェーススタックに記憶する。

（３）引数リンケージ

引数データについては、インタプリタはスタック渡し、コンパイラはレジスタ渡しのため、各々のコードに移行する際に、引数リンケージの差を吸収した。

（４）カット処理

カット範囲が別コードを含む場合、例えばＣコードのバックトラック時に、カット組み込み述語によってカットされる述語にＩコードがある場合のバックトラックをインタフェース・カット・スタックを設けることによって実現した。

図３では、コンパイラソース内の述語ｃのｆａｉｌによるバックトラック時に、述語ｂがカットされるときのスタック状態を示している。（１）でＩコードへのバックトラックのために、インタプリタローカルスタックのアドレス(FP)を記憶すると述べたが、無条件にｐｏｐするとインタプリタは一番近い別解ｂをバックトラックさせてしまう。そこで、FPとバックトラックフレームベースレジスタ(BFBR)をインタフェース・カット・スタックにペアで記憶し、Ｉコードへのバックトラック時に、記憶したBFBRと現在のBFBRを比較することで、ペアで記憶したFPが有効か否かを判断できるようにした。

FP有効：　記憶したBFBR ≧ 現在のBFBR
FP無効：　記憶したBFBR ＜ 現在のBFBR

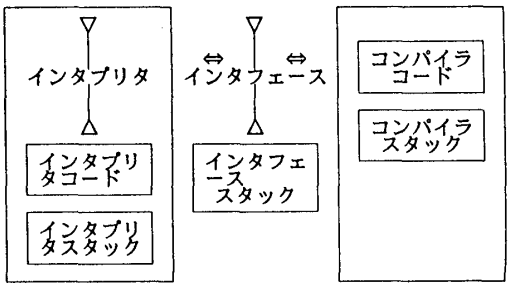

図２　インタフェース処理の概念

High Performance Integrated Prolog Processor IPP (XII) System Integration
Hiroshi Suzuki 1), Yoshiyuki Otsu 1), Takashi Ishikawa 1), Sigeo Abe 2)
1) Hitachi Process Computer Engineering ,Inc.
2) Hitachi ,Ltd.

図３では、カットによって、現在のBFBRが#1でなく、#2であるので、インタフェース・カット・スタックの上位にあるFPは無効となり、結果的に#3をカットし、#4をバックトラックすることができる。

ここでは、Cコード内でのカットについてしめしたが、Iコード内でのカットについても同様に実現した。

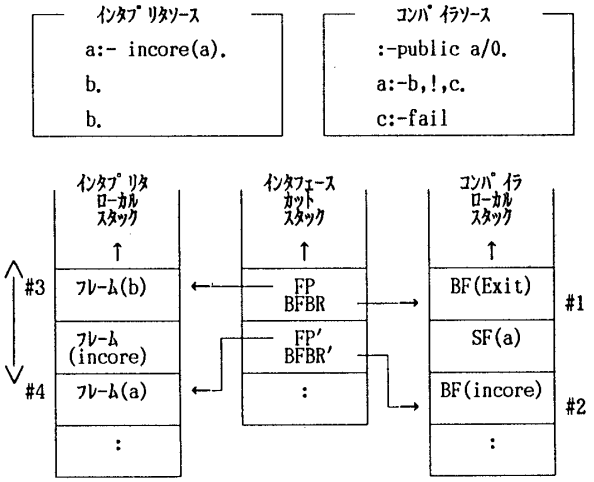

図3　インタフェースのカット処理

4．性能測定結果

本ＰＲＯＬＯＧシステムの性能について、コンパイル速度、オブジェクト効率、インタプリタとコンパイラコードの実行速度を測定した。性能測定には、第１回ＰＲＯＬＯＧコンテスト3)で用いられたベンチマークプログラムを使用した。

（１）コンパイル速度

表１に、コンパイル時間の全測定結果を示す。

総合して、７０７０行／分である。

（２）オブジェクト効率

表１に、オブジェクト容量の全測定結果を示す。

総合して、４８ｂｙｔｅ／行である。

表１　コンパイル速度とオブジェクト効率

No.	ﾌﾟﾛｸﾞﾗﾑ名	ｿｰｽ行数 (step)	ｺﾝﾊﾟｲﾙ時間 (sec)	ｵﾌﾞｼﾞｪｸﾄ容量 (kbyte)
1	bench1	759	4.7	19.5
2	bench2	691	5.0	23.7
3	bench3	113	0.9	3.2
4	bench4	131	0.8	2.6
5	bench5	89	1.0	3.9
6	bench6	187	1.3	4.6
7	bench7	303	3.2	14.2
8	bench8	153	1.2	3.6
9	bench9	479	4.8	27.7
10	bench10	1528	14.4	93.3
11	bench11	1528	14.7	100.4
12	bench12	110	0.9	2.8
計		6233	52.9	299.5
効率		-	7070 行/分	48 byte/行

（３）実行速度

表２に、インタプリタとコンパイラコードの実行速度の全測定結果を示す。

表２　ベンチマーク実行速度

No	ﾍﾞﾝﾁﾏｰｸ	msec	
		ｲﾝﾀﾌﾟﾘﾀｺｰﾄﾞ	ｺﾝﾊﾟｲﾙｺｰﾄﾞ
1	Atom-1	5.18e-2	2.93e-3
2	Atom-5	9.90e-2	4.00e-3
3	Var-1	7.83e-2	3.36e-3
4	Var-5	1.68e-1	5.74e-3
5	Con-1	8.10e-2	4.03e-3
6	Con-5	2.17e-1	7.67e-3
7	Str-1	9.47e-2	3.90e-3
8	Str-5	2.52e-1	9.01e-3
9	Str-Var-1	8.07e-2	4.37e-3
10	Str-Var-5	2.13e-1	8.50e-3
11	Dat-Call	1.48e-1	3.94e-3
12	Ndat-Call	2.60e-1	6.89e-3
13	Shallow-3	7.81e-1	2.18e-2
14	Deep-Back	2.12	5.92e-2
15	Key-First	7.53e-2	3.14e-3
16	First	7.00e-2	5.22e-3
17	Key-Last	5.60	3.05e-3
18	Last	7.81	3.78e-3
19	Key-Midd	2.82	3.26e-3
20	Nrev-30	8.84e+1	4.14e-1
21	Sort-50	1.16e+2	1.05
22	Cons-1000	4.12e+1	4.60e-1
23	Trav-1000	1.63e+1	3.76e-1
24	Srev-4	2.80e+1	1.31
25	Srev-5	1.12e+2	5.85
26	Srev-6	4.55e+2	2.4e+1
27	Lisp-Tara	2.03e+4	4.27e+2
28	Lisp-Fib	——	——
29	Lisp-Rev	8.39e+2	1.90e+2
30	S-Queen-1	8.50e+2	5.40
31	S-Queen-a	1.36e+3	8.60e+1

5．おわりに

本稿では、インタプリタとコンパイラとを統合したＰＲＯＬＯＧシステムの開発し、その処理方式と性能を報告した。

今後は、インタプリタコードととコンパイラコードのインタフェースオーバヘッドの削減を検討してゆきたい。

参考文献

1) D.Warren, "An Abstract Prolog Instruction Set", Technical Note 309,AI center,SRI,1983.
2) 坂東他、" 内蔵型高速PROLOGプロセッサIPP(Ｉ)－(ＶＩ)"、第34情報処理全国大会、昭和62年3月.
3) 奥乃、" 第３回Lispコンテスト及び第１回Prolog コンテスト報告"、記号処理33-4、昭和60年.