PROLOG 処理系の動的 CDR コーディング方式

碓崎 賢一[†]

PROLOG は他の言語処理系と比較してメモリ使用量が多いという問題がある．また，PROLOG の基本操作である単一化処理はメモリのアクセス頻度が高いために，RISC 技術などによるプロセッサの高速化の一方で，キャッシュミスによるメモリアクセスの遅延が，処理速度を抑制する大きな要因となりつつある．本論文では，PROLOG 処理系においてリストの内部表現のメモリ効率を2倍に高めるとともに，メモリの参照頻度を低下させる動的 CDR コーディング方式を提案し，その性能評価を示す．本方式は，CDR コーディングの可能性を動的に検査するために，リストの多くの要素を CDR コーディングすることができる．CDR コーディングのための処理項目の増加による速度低下は，RISC プロセッサの遅延スロットを利用して抑えている．CDR コーディング方式では，従来の CONS による方式と比較して，メモリ参照の局所性が高くなりキャッシュミスの頻度を小さく抑えられるために，データサイズの大きな領域では従来方式を越える性能が得られることが明らかになった．

Dynamic CDR Coding Method for PROLOG Systems

KEN'ICHI KAKIZAKI[†]

This paper describes dynamic CDR coding method for PROLOG systems. This method improves memory utilization efficiency for list expressions by two times, and reduces the memory reference ratio. Because possibilities of the dynamic CDR coding are dynamically examined, large amount of the list elements are CDR-coded. The processing speed of the system using CDR coding method is generally slower than that of conventional method, however, this method realizes performance that is comparable to conventional method by using delay slots on the RISC processors. CDR coding method improves locality of memory references, and it prevents increase of cache miss rate on large data processing. The low cache miss rate provides performance advantage compared with the conventional method.

1. はじめに

PROLOG は単一化と後戻りを基本とした柔軟で強力な表現力と処理能力を持つが，他の言語と比較して大容量のメモリを消費するという問題を持つ．PROLOGで多用されるデータ型としてリストがあり，リストを圧縮表現することができればメモリの使用量を低減させることができる．本論文では，リストを圧縮表現することにより PROLOG 処理系のメモリ効率を改善させる動的 CDR コーディング[1)-3)]の実現方式を提案しその性能評価を示す．

CDR コーディング方式は，メモリ効率は改善されるが，そのオーバヘッドのために，汎用プロセッサ上では処理速度が低下するという問題がある．提案方式では，最近多用されている RISC のロード，分岐命令の遅延スロットを有効に利用することで，CDR コーディング処理にともなうオーバヘッドを低減させている．評価の結果，大容量のデータを扱うプログラムでは，CDR コーディング方式のほうが従来方式よりも高速であることが明らかになった．

近年，半導体メモリの大容量化が進んでおり，メモリ効率の観点だけでは CDR コーディングの価値は低いと考えられる．一方，RISC に代表される技術によりプロセッサの処理速度が急速に向上したが，キャッシュミスのペナルティを原因とするシステムの実効性能の頭打ちが顕在化しつつあり，デスクトップシステムにおいてもキャッシュメモリの階層化が進展しつつある．このような状況では，メモリの参照頻度を低下させ，プロセッサ内部だけ，あるいはキャッシュメモリも含めたプロセッサユニットだけで処理を継続する処理方式が重要となってくる．本論文で提案する動的 CDR コーディング方式は，このような流れに沿った実現方式である．

† 九州工業大学情報工学部電子情報工学教室
Department of Computer Science and Electronics, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology.

2. CDR コーディング

CDR コーディング手法は，リストの CDR 側にリストが連続することが非常に多いという特性を利用したもので，**図1**に示すように，リストの要素を連続領域に配列状に格納して CONS セルの CDR 側のポインタを省略することにより，使用するメモリ量を2分の1に削減することができるという利点がある．しかしながら，従来の CDR コーディング方式は，処理のオーバヘッドが大きかったために，PROLOG マシンなどの専用機以外で使用されることはなかった．

2.1 静的な CDR コーディング

PROLOG の CDR コーディング方式として，PROLOG マシン PLM-1 に実装されている方式[4]がある．PLM-1 の方式は，静的な解析で連続領域に作成できることが判定できるリストのみに対して CDR コーディングを行うものである．たとえば，**図2**に示すゴールmakecons/1 を起動すると，リスト [a, b, c, d] が作成され変数 X に束縛される．このリストのうち makelist/1 によって作成される [b, c, d] の部分は，連続領域に格納できることがコンパイル時に検出できるために CDR コーディングされる．しかしながら，makecons/1 によって作成されるリストの最初の要素は，CDR 側の値が連続領域に作成されるか否かがコンパイル時に決定できないために，通常の CONS セルとして作成される．このように，静的な解析による CDR コーディング方式では，リストの要素が連続領域に格納できる場合（図2.2）であっても，そのすべての要素を CDR コーディングすることができない（図2.1）という問題がある．

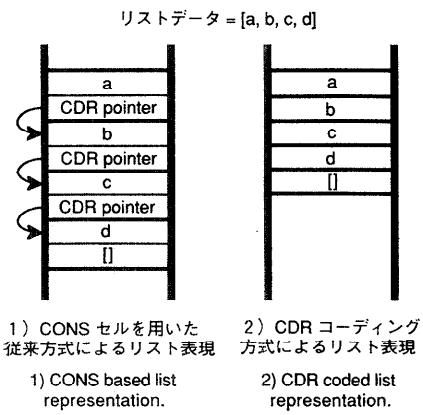


図 1 CDR コーディング
Fig. 1 CDR coding.

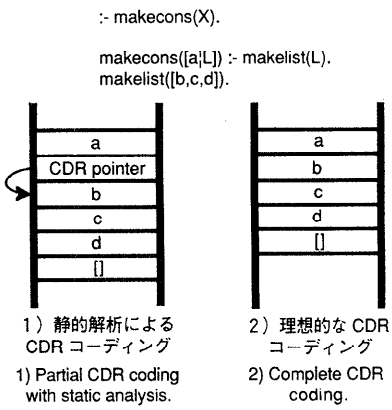


図 2 静的な解析による CDR コーディング
Fig. 2 CDR coding with static analysis.

上記の問題点が明確に現れるのは，append/3 のようにリストの要素を1つずつ作成する場合である．このような操作はリスト処理の基本として多用されているが，静的な解析による CDR コーディング方式では，可能であるにもかかわらず，まったく CDR コーディングされないという問題が生じる．

2.2 動的な CDR コーディング

本論文で提案する動的 CDR コーディング方式は，2.1節で示した従来方式の問題点を解決するもので，次のような特長を持つ．

1) 特別な述語などによらず通常のリストの単一化処理により CDR コーディングが行える．
2) CDR コーディングの可能性を実行時に動的に検査する．
3) RISC などの汎用プロセッサ上でも CDR コーディングのオーバヘッドが小さい．

これらの特長により，一般のコンピュータシステムで使用できるだけでなく，append/3 などのリストを1つずつ作成する述語においても CDR コーディングが活用される効果的な実現方式となっている．この実現方式は，WAM[5] に基づく PROLOG 処理系の次のような基本特性を有効に用いている．

A）データ領域がスタック上に連続して確保される．
B）リストが頭部から作成される．
C）リストの破壊的操作がない．

A，Bの特性により，特別な操作を施すことなく，CDR コーディングに必要な連続領域を確保した上で，CDR コーディングされたリストを段階的に作成することが可能になっている．処理速度の点では，Cの特性が大きく貢献しており，単純でオーバヘッドの少な

い処理を行えるようになっている．

リストの破壊的操作がある LISP の CDR コーディング[6]では，リストのセルは次の4種類に分類され，CDR コーディング用の2ビットのタグで表されている．

1) 通常の CONS セル
2) CDR コーディングされたセル
3) 破壊的操作を受け間接ポインタを持つセル
4) NIL で終端しているセル

CDR コーディングで本質的に必要な分類は1，2の2種類だけであるが，LISP では RPLACD などのリストの破壊的な操作が行えるために，3の分類が必須になる．4の分類は，2ビットで表される4状態の残りの1つを有効に利用するために導入されたものと考えられる．

CDR コーディングを含むリストを参照するためには，実行時に上記の4種類のリストの状態を判断し，それぞれに適切な処理を行う必要がある．LISP マシンなどの専用機では，タグによる4方向の分岐を1クロックで実行したり，その処理を他の処理と並列に行うことができるために，CDR コーディングのオーバヘッドを除去することができる．しかしながら汎用のプロセッサでは，そのようなタグを効率よく処理するための特別な機能が付加されていないために，2ビットのタグに対する最低2回の条件判断を必要とする CDR コーディング処理は大きなオーバヘッドを生じることになり，処理速度の問題で一般的に使用されることがなかった．

PROLOG では，リストの破壊的な操作がないために3の分類は必要なく，4の付加的な分類も削除できるので，1ビットのタグに対する1，2の分類を1回の条件判断だけで行うことができる．さらに，この1回の条件判断のオーバヘッドのほとんどは，RISC の遅延スロットを利用することにより除去することができるために，一般的な RISC プロセッサで効率よく CDR コーディングされたリストを参照できることになる．このように，専用マシンだけで CDR コーディングが実用化されている LISP よりも，PROLOG のほうが CDR コーディングに適した特性をもっていると考えられる．

3. 動的 CDR コーディングの処理方式

3.1 タグとレジスタの追加

CDR コーディングを実現するために，WAM 方式に以下の拡張を施す．

1) データ表現内に1ビット (CDR-TAG) を確保
2) CDR コーディング用ポインタ (CDR-P) を追加
3) CAR レジスタ (CAR-REG) を追加
4) 選択点の構造に CDR-P と CAR-REG を追加

1は各リストが CDR コーディングされているか否かを示すために用い，2はリストを作成する場合に CDR コーディングが可能か否かを検査するために用いる．3は CAR セルの値を保存し，メモリのアクセス頻度を減少させるために利用するもので，必須ではないが性能向上のために重要である．4は再試行を行う際に，CDR コーディング用の情報を含めて実行環境を復元するために用いる．

3.2 単一化処理

3.2.1 リストの参照

リストの CAR セルの値を得る処理では，まずそのリストの CAR セルの CDR-TAG を検査し，CDR コーディングが行われているか否かを判定する．CDR コーディングが行われており，CAR セルの値を使用する場合には，CDR-TAG を取り除く処理を行う．リストの CDR セルの値を得る処理では，CDR コーディングが行われている場合（**図 3.1**）には，リストの CDR セルのアドレスを CDR 側の値とする．CDR コーディングが行われていない場合（図 3.2）には，従来のリスト処理と同様に，CDR セルの内容を CDR 側の値とする．入力モードに指定された append/3 の第1引数を例として，**図 4** に処理方法を示す．

3.2.2 リストの生成

この項では，CDR コーディングされたリストを動的に生成する手法を示す．なお，静的な解析によって，コンパイル時に CDR コーディングが可能なことが判定できる場合の処理は特に示さないが，PLM-1 と同様の処理を行う．

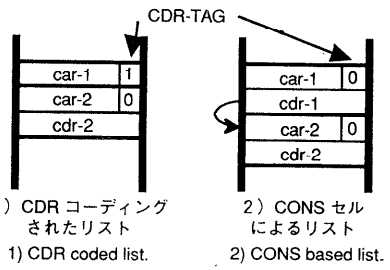


図 3 リストの参照
Fig. 3 Read mode for list.

```
/*      :- mode append(+, +, -).
        append([X|L1],L2,[X|L3]) :- append(L1,L2,L3). */

typedef unsigned int UINT ;
typedef UINT TERM, *TERMP ;
typedef struct {
        TERM car ;
        TERM cdr ;
} LIST, *LISTP ;

#define CDR_TAG      0x00000001
/* ------------------------------------------- */

register LISTP cons ;
register TERM Arg[MAX_ARG], p, tmp3 ;

p = Arg[0] ;      /* 第1引数の値を取得する */
Deref(p) ;        /* デリファレンスを行う */

cons = LAddr(p) ;       /* データ型を示すタグを取り除く */
if ((UINT)cons->car & CDR_TAG) {
        tmp3 = (TERM)((UINT)cons->car & ~CDR_TAG) ;
        Arg[0] = (TERM)&((LISTP)p)->cdr ;
} else {
        tmp3 = cons->car ;
        Arg[0] = cons->cdr ;
}
```

図 4 リストの参照処理
Fig. 4 Read mode code for CDR coding.

append/3 の第3引数のように1つずつリストを生成する場合には，CDR コーディングが可能か否かの検査を動的に行う．**図 5.1** にリストが作成された直後のヒープの状態を示す．CDR-P には最後に作成されたリストの CDR セルのアドレスが保存され，新たにリストが作成されるたびに更新される．HP はヒープの空き領域を指すレジスタで，新たなデータ構造はこのアドレス以降に作成される．CDR-P を操作するのはリストを生成する処理だけであり，複合項などが作成された場合には，図 5.2 に示すように HP のみが進められ，CDR-P は変更を受けない．

新たに作成するリストが CDR コーディング可能か否かは，上記の2つのレジスタの値と新たに生成されるリストを束縛する変数アドレスの三者の関係と，直

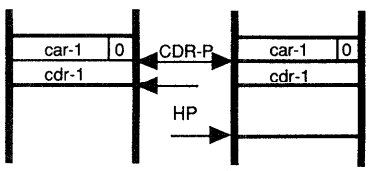


図 5 CDR コーディングの可能性
Fig. 5 CDR coding feasibility.

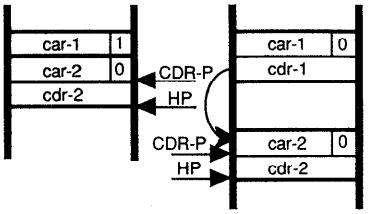


図 6 リストの生成
Fig. 6 Write mode for list.

前に作成されたリストの CAR セルが変数か否かで判定される．

新たに作成するリストが CDR-P で指されているリストの CDR セルに代入され，HP が CDR-P で示されているリストに連続した領域を指している場合には，リストを連続領域に作成することができるので CDR コーディングを行う．この場合には，CDR-P が指していたアドレスに新たなリストを作成するとともに，**図 6.1** に示されるように CDR-TAG を前のリストの CAR セルに設定して，CDR-P と HP をそれぞれ CONS セルの半分だけ進める．この操作により，新たに作成されるリストが，すでに作成されているリストの CDR 側に埋め込まれ CDR コーディングが行われる．

上記の条件が満たされている場合でも，直前に作成されたリストの CAR セルが変数の場合には CDR コーディングを行わない．これは，CDR-TAG が付加された変数が存在することになると，デリファレンス処理やすべての変数の単一化処理で CDR-TAG の検査を行わなければならなくなり，オーバヘッドが非常に大きくなるためである．リストを作成する処理では，具体化されていない変数が CAR セルに来ることは非常に希であるため，この制約は CDR コーディングされる要素の数にはほとんど影響しないと考えられる．

CDR コーディングを行う際に，後戻りに備えて前のリストの CDR セルのアドレスをトレイルに記録する必要がある場合には，CDR セルのアドレスに CDR-TAG を付加してトレイルスタックに記録する．後戻りでトレイル情報を用いて変数の束縛を解いていく処理では，CDR-TAG が設定されているポインタに対しては，変数を未定義にするだけではなく，その前のセル（リストの CAR セル）に付加されている CDR-

```
register TERMP hp, cdrp ;
register TERM car_reg ;

p = Arg[2] ;      /* 第1引数の値を取得する */
Deref(p) ;        /* デリファレンスを行う */

if (cdrp !=p || (cdrp + 1) != hp || IsRef(*((TERMP)p - 1))) {
        /* CONS の作成 */
        cons = (LISTP)hp ;
        Trail(p) ;        /* 通常のトレイルの記録 */
        *(TERMP)p = SetTag(cons, LIST_TAG) ;
        cdrp = hp + 1 ;
        hp += 2 ;
} else {
        /* CDR コーディング処理 */
        cons = (LISTP)p ;
        cdrTrail(p) ;     /* CDR Coding 用のトレイルの記録 */
        *((TERMP)p - 1) =
                (TERM)((UINT)*((TERMP)p - 1) | CDR_TAG) ;
        cdrp++ ;
        hp++ ;
}

cons->car = tmp3 ;

Arg[2] = cons->cdr = (TERM)&cons->cdr ;
```

図 7 リストの生成処理
Fig. 7 Write mode code for CDR coding.

TAG もあわせて除去する.

CDR コーディングを行えない場合には，CONS による通常のリスト（図 6.2）を作成し，新たなリストの CDR セルのアドレスを CDR-P に設定する.

出力モードに指定された append/3 の第 3 引数を例として，上記の処理の概要を**図 7** に示す．なお，選択点を生成する場合には後戻りに備えて，CDR コーディングで使用する CDR-P, CAR-REG を選択点に保存する処理を行う．後戻り時には，これらの情報を含めて環境を復元をする.

3.2.3 複数リストの生成

3.2.2 項では，1つのリストが作成される append/3 の例を示したが，本提案方式では，複数のリストが作成される**図 8** の partition/4 のような述語でも，1つの系列のリストが連続して作成される場合には動的な検査によって CDR コーディングが行われる．図 8 の例では，1，2 のゴールを実行すると，X には [1, 2, 3]，Y には [10, 11, 12] が作成される．1 の例では，2 組のリストがそれぞれ連続して作成され 3.2.2 項に示した条件が満たされるために，CDR コーディングされる．2 の例では，2 組のリストは交互に作成され条件が満たされないために，通常の CONS によるリストとなる．本方式では実行時の検査によって CDR

```
:- partition([1, 2, 3, 10, 11, 12], 7, X, Y).
    1) CDR コーディング可能
    1) CDR coding is possible.
:- partition([1, 10, 2, 11, 3, 12], 7, X, Y).
    2) CDR コーディング不可能
    2) CDR coding is impossible.

partition([X| L1], Y, [X| L2], L3) :-
        X =< Y,
        !,
        partition(L1, Y, L2, L3).
partition([X| L1], Y, L2, [X| L3]) :-
        partition(L1, Y, L2, L3).
partition([], _, [], []).
```

図 8 複数リストの CDR コーディング
Fig. 8 CDR coding for multiple lists.

コーディングが行われるために，複数のリストを作成するプログラムでは，処理されるデータの特性によって CDR コーディングの成功率が変動する.

3.2.4 汎用単一化ルーチン

ヒープ上に作成された項と項との汎用の単一化処理では，リストの要素を並べて新たなリストが作成されることはない．したがって，リストの参照（分解）処理のみを 3.2.1 項に示した方法で CDR コーディングに対応させればよい.

3.3 RISC の特性を利用した最適化

3.2 節で示した処理方式をそのまま利用すると，動的な検査処理の追加により実行速度が低下してしまうという問題がある．このような問題を解決するために，ワークステーションで広く使用されている RISC プロセッサの，メモリの参照コストが高いという特性と，分岐によって生じたパイプラインの空きスロットを利用して命令を実行できるという特徴を利用して，性能の低下を抑制する方法を示す.

3.3.1 リストの参照

RISC のメモリからレジスタにロードする命令では，ロードされた値を使用できるまでに 1～2 クロックの遅延が必要となる．したがって，図 4 の CAR セルの CDR-TAG の検査のように，メモリからロードした値を使用する処理では，プロセッサに待ち状態が発生する．そこで，このような待ち時間を有効に使用できるように，デリファレンス以降の処理を**図 9** に示すように変更し最適化する．図 9 では，CAR セルのロードの待ち時間を利用して CDR セルのアドレスを求めており，リストが CDR コーディングされていた場合の CDR 側の処理時間を実質的に除去している.

図 9 では，CDR コーディングされている場合に，CAR セルの値から CDR-TAG を除去する処理を行

```
cons = LAddr(p) ;
tmp3 = cons->car ;
p = (TERM)&((LISTP)p)->cdr ;

if ((UINT)tmp3 & CDR_TAG) {
        tmp3 = (TERM)((UINT)tmp3 & ~CDR_TAG) ;
} else {
        p = cons->cdr ;
}

Arg[0] = p ;
```

図 9 リストの参照処理の最適化
Fig. 9 Optimized read mode code for CDR coding.

っているが，この処理は，CDR コーディングされていないリストの CAR セルの値に対して行っても影響がない．したがって，CDR コーディングされているか否かの条件分岐の遅延スロットで実行することにより，処理時間を実質的に除去することができる．

RISC プロセッサ MC 88100[7]を例として，このように最適化された図 9 の処理をアセンブリ言語で**図 10** に示す．図 10 のアセンブリコードの右端に，処理に要する実効クロック数を示している．図の nop は，ロードによって読み込まれた値が，継続する命令で利用できるようになるまで，同期をとるために自動的に挿入されるもので，88100 のロードでは 2 クロックの

```
; r22 = p, r14 = 0x0fffffff, r17 = temp, r18 = Arg
; r12, r15 は作業用レジスタ
        and             r15,r14,r22                     ;1
        ld              r17,r0,r15                      ;2
        ld              r12,r15,4                       ;3
;       nop     ;r12 に値がロードされるまでの待ち状態   ;4
;       nop     ;同上                                   ;5
        st              r12,r0,r18                      ;6
```

1）従来方式による CONS セルのリストの参照
1) Clock cycles for CONS based list.

```
        and             r15,r14,r22                     ;1
        ld              r17,r0,r15                      ;2
        addu            r12,r14,4                       ;3
;       nop     ;r17 に値がロードされるまでの待ち状態   ;4
        bb1.n           0,r17,@L8                       ;5
        and             r17,r17,lo16(0xfffffffe)        ;6
        ld              r12,r15,4
;       nop     ;r12 に値がロードされるまでの待ち状態
;       nop     ;同上
@L8:
        st              r12,r0,r18                      ;7
```

2）CDR コーディングされたリストの参照
2) Clock cycles for CDR coded list.

図 10 リストの参照の実効クロック数
Fig. 10 Effective execution clock cycles for read mode.

遅延が生じる．ロードや分岐の遅延スロットを有効に利用することにより，実行時間は 6 クロックから 7 クロックの 17% ほどの増加に抑えられている．述語単位の処理では，この他にデリファレンスやインデキシングなどの共通の処理があり，処理時間の増加率を算出する処理時間の分母が大きくなるために，実質的な処理時間の増加は数%に抑えられる．

3.3.2 リストの生成

図 7 の処理では，アドレス p-1 で示される直前のリストの CAR セルの値をメモリから参照するために，処理時間が増加する問題がある．この問題は，3.1 節に示した CAR-REG を利用することによって解決する．図 7 のデリファレンス以降を**図 11** に示すように最適化し，直前に作成したリストの CAR セルの値を CAR-REG に保存しておくことにより，コストの高いメモリ参照を回避することができる．

図 11 の CDR コーディングが可能か否かの検査では，図に示した（1），（2），（3）の各検査の条件分岐の遅延スロットで以下のような操作を行うことによって，CDR コーディング処理のコストの低減をはかる．

（1） CDR-P と引数の比較では，インクリメント演算子で示されているように CDR-P を増加させることにより，CDR コーディングが行える場合の CDR-P の更新処理を行う．

（2） CDR-P と HP の比較では，（3）で行う CAR-REG のデータ型の検査に備えて，そのタグを取得する処理を行う．

（3） CAR-REGが変数か否かの検査では，CDRコーディング処理に分岐した場合に行われる cons＝p の代入処理を行う．

```
if (cdrp++ != p || cdrp != hp || IsRef(car_reg)) {
/*      (1)         (2)         (3)     */
        cons = (LISTP)hp ;
        Trail(p) ;          /* 必要性に応じてトレイルを記録 */
        *(TERMP)p = SetTag(cons, LIST_TAG) ;
        cdrp = hp + 1 ;
        hp += 2 ;
} else {
        cons = (LISTP)p ;
        cdrTrail(p)         ;/* 必要性に応じてトレイルを記録 */
        *((TERMP)p - 1) = car_reg ;
        hp++ ;
}

cons->car = tmp3 ;
car_reg = (TERM)((UINT)tmp3 | CDR_TAG) ;

Arg[2] = cons->cdr = (TERM)&cons->cdr ;
```

図 11 リストの生成処理の最適化
Fig. 11 Optimized write mode code for CDR coding.

変数のトレイル処理は，従来方式では，変数がヒープにあるのかローカルスタックにあるのかを検査し，次に，それぞれの領域で選択点が作成された前か後かを検査する2段階の処理が必要であった．また，RISC のレジスタ数の制約などにより，ヒープとローカルスタックの境界アドレスをレジスタ上に置けずメモリに格納している場合には，ロードによる待ち状態が発生するために，検査時間が長くなるという問題がある．一方，本方式では，CDR コーディングが行えるということが判定された時点で変数がヒープにあることが自明となり，トレイルの判定は一度だけの検査で行えるために，CDR コーディングの条件検査などで増加する処理の多くを打ち消すことができる．

3.4 抽象 PROLOG マシンの命令セット

3.3節の図10に示したコードは，モード宣言を付加した状態での WAM の get_list と unify_variable 2個の3命令の組み合せに相当する．3.3節に示したコードを WAM の命令の単位に分割することは可能であるが，特に入出力両用で処理内容の多い一般的な命令では，一連の処理として RISC の特性を利用した十分な最適化を施せなくなる．また，十分な最適化を行おうとすると，WAM コードから RISC のマシンコードへの翻訳系で，各 RISC プロセッサの特性にあわせたマシンコードレベルの命令のスケジューリング機能が必要となり複雑になるという問題がある．

CDR コーディング用の命令としては，上記の理由により，get_list, unify_variable, unify_value を組み合わせた複合命令を用いる．例えば，図9に対応する命令は get_list_var_var，図11に対応する命令は get_list_val_var とする．これらの命令はリストの要素が変数の場合に対応したものであるが，リストの要素が変数以外の場合も，それぞれのデータ型に対応させた個別の命令を導入するのではなく，上記の命令と get 系の命令を組み合わせて処理を行い，命令数の増加を抑制する．実際の処理では，CAR や CDR の値をまず一時変数レジスタ上にコピーし，その値に対して定数項の単一化処理を行う．例えば，[123|*X*]に対応する命令は以下のようになる．

```
get_list_var_var    Arg[0], temp1, temp2
get_constant        123, temp1
```

専用の命令を設けると比較して WAM コードレベルでは冗長な処理になっているように見えるが，RISC プロセッサのレジスタのいくつかを一時変数レジスタとして使用し，上記の temp1 をそのレジスタに割り当てると，定数項専用の命令を導入したのと処理速度は変らない．これは，RISC で比較などの処理を行う場合に，値を一度レジスタに読み込まなければならないためで，RISC のレジスタを一時変数レジスタに割り当てると，1つの専用命令でデータを内部的にレジスタに読み込むか，2つの汎用命令で明示的にレジスタに読み込むかの違いが生じるだけで，結果的に同一の処理を行うことになるからである．

4. 評価と考察

4.1 評価内容

CDR コーディング方式の性能評価を行うために，4種類の RISC マシンで append/3 を使ったベンチマークテストを行った．測定はワークステーションを占有した状態で行い，同一のデータ領域を使用して append/3 を繰り返し実行させ，処理を行うリストの長さとCPU時間に基づく処理速度との関係を調べた．ベンチマークテストの結果を**図12**に示す．

図12の○，□，△の系列は3.3節に示したように最適化された CDR コーディング用の append/3，×の系列は従来の方式による append/3 を使用して測定したもので，両者ともモード宣言が付加されており，最適化の程度と CDR コーディングに直接関係ない部分は共通である．各コードは，WAM 方式に基づいてC言語で記述されており，スタックポインタ，一時変数レジスタ，CDR コーディングで導入したレジスタをプロセッサのレジスタに割り当て，引数レジスタとスタック領域の境界を示すアドレスはメモリ上に割り当てている．これらのコードは，最適化Cコンパイラによってコンパイルされており，従来方式のリスト処理のコードも含めて，可能な部分は遅延スロットが埋められている．

△は基本的には○，□と同じコードであるが，CDR コーディングが可能な場合であってもそのように判断されないように，CDR-P の更新処理を変更している．○と△の入力には CDR コーディングされたリストを，□と×には通常の CONS によるリストを与えた．図12の横軸は使用したリストの長さを示している．測定では与えられたリストと生成されたリストの組によってメモリが使用され，図の左端に示されるリストの長さが100の場合には，従来方式では1.6Kバイト，CDR コーディング方式では 0.8K バイトが使用されることになる．

評価に append/3 を選択した理由は，その処理がリ

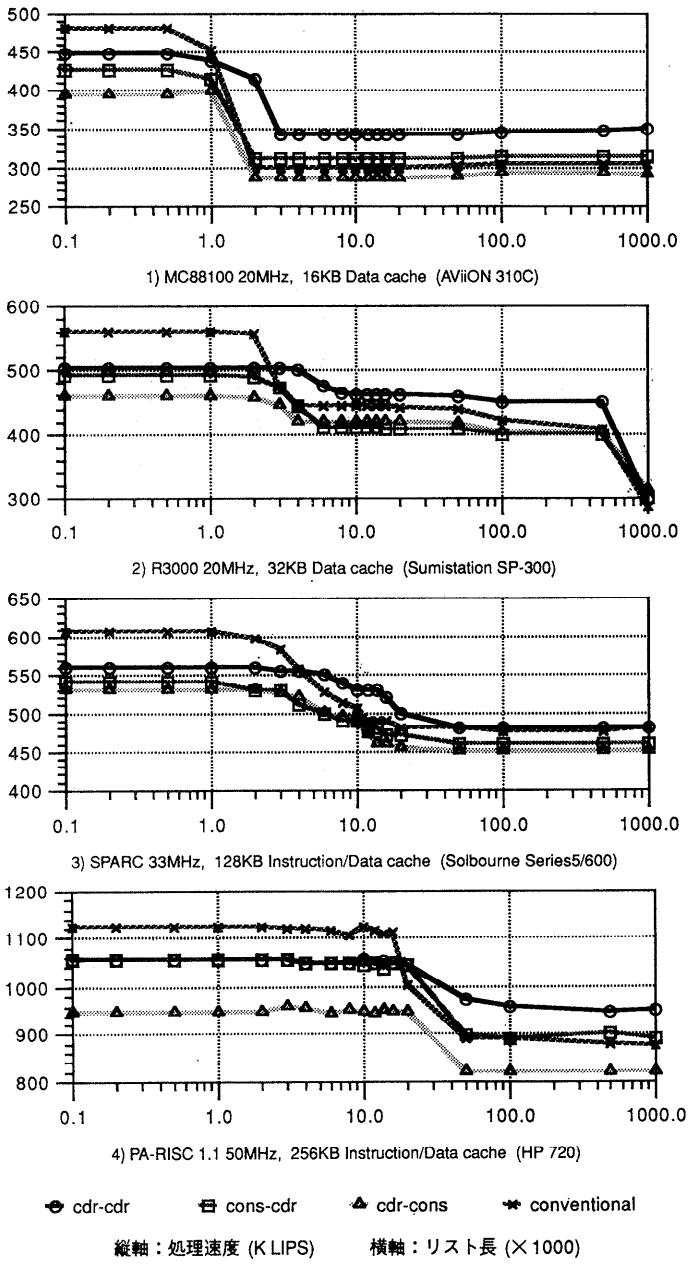


図 12 性能評価
Fig. 12 Performance evaluation for CDR coding method.

ストの分解と生成というリスト処理の基本操作を1つずつ含む一方で，他の操作をほとんど含まないためにCDR コーディングの重点的な性能評価に適切であるためである．一般的な述語では，リスト以外の単一化処理や，選択点や環境の操作が増加して，リスト処理の比率が低下する．したがって，この評価による CDR コーディングの性能が従来方式と比較して高い場合でも低い場合でも，一般的な状況における性能比は，この評価で示される性能比以下になると考えられる．

4.2 処理速度の変化特性

図12では，リストが短い場合には従来の方式と比較して CDR コーディング方式のほうが遅いが，リストが長くなると両者の処理速度が低下すると共に，CDR コーディング方式のほうが速くなることが示されている．○と×の2系列の性能が低下しはじめる点は，リストの長さで2倍ほど異なっており，従来の方式では急速に性能が低下するのに対して，本方式では緩やかに低下するという違いが生じている．リストが長くなると処理速度が低下するのは，キャッシュのヒット率が低下するためであると考えられる．CDR コーディング方式では，リストの長さが同じ場合，従来方式の半分のメモリしか使用しないために，キャッシュミスの影響が少なく，性能低下の割合が低くなっている．

CDR コーディングの効果として，リストを圧縮表現しワーキングセットを局所化することができるために，ページング頻度やキャッシュのミス率を低減できることが一般に知られている．ページング頻度の減少に関しては，メモリとディスクのアクセス速度が数桁のオーダで異なるためにその効果がよく知られている．一方，キャッシュのミス率の低減による効果については，従来の CDR コーディングでは，処理のオーバヘッドの増加による速度低下のほうが大きいために，議論されることはなかった．本論文で提案する動的 CDR コーディング方式は，処理のオーバヘッドが小さく，一般的なリスト処理方式とほぼ同等の処理速度が得られるという利点がある．また，大容量のデータを用いる場合には，一般的なリスト処理方式よりもキャッシュの

ミス率が低減されるために処理速度の低下が少なく，その結果として従来のリスト処理方式よりも処理速度が上回ることになる．

4.3 性 能 比

図 12 の○と×で示される CDR コーディング方式と従来方式の性能比を**図 13** に示す．図 13 では，リストが短い場合には CDR コーディング方式のほうが 8 %程度遅いが，リストがある程度長くなると，比率が大きなもので 40% ほど速くなるピークが示されている．これは，従来方式ではリストがキャッシュに入り切れなくなる一方で，CDR コーディング方式ではリストがすべてキャッシュに入っている状態であると考えられる．また，各マシン間で，従来方式と CDR コーディング方式の性能が反転する点と両者の性能比がピークになる点は，ほぼそれぞれのデータキャッシュ容量に比例している．現状では，RISC に代表されるプロセッサの大幅な高速化により，プロセッサの処理速度と主メモリのアクセス速度との格差が広がりつつあるため，上記のピーク時の性能比やリストが長い場合の性能比は，CDR コーディングに有利な方向にさらに大きくなっていくものと考えられる．

CDR コーディングに不利なリストが短い場合の性能低下は 8 %ほどに抑えられているが，モード宣言を付けていない一般的なコードでは，入出力モードの判定などの実行時の処理が増加するために，性能比は改善されさらに 1 に近くなると考えられる．また，ワークステーションでは，個人で使用している場合でも X ウィンドウサーバをはじめメモリを多量に消費するプログラムが実行されていたり，複数の利用者が様々なプログラムを実行させるために，キャッシュミスの影響が現れるのはこの評価での数分の一のリストの長さからであると考えられる．さらに，PLM-1 などで採用されている静的な CDR コーディング手法を併用すると，静的に CDR コーディングできることが判明したリストの生成は，通常の CONS によるリストの半分の処理時間ですむ．このため，動的 CDR コーディング方式を採用したシステムでは，メモリ効率を向上できるだけでなく，一般的な使用状況で従来のシステムを上回る性能を実現できるものと考えられる．

4.4 CDR コーディングの成功率

図 12 では，同じ CDR コーディング用のコードであっても，入出力のリストが共に CDR コーディングされている○の系列の性能が，他の 2 者と比較して高いことが示されている．これは 3.3 節で示したように，CDR コーディングに対してコードの最適化を行っているためで，分岐命令の遅延スロットの使い方を CONS を使用する側に最適化すれば，この関係を逆転できる．したがって，述語をコンパイルする場合に，生成されるリストが CDR コーディングされる頻度が高いか否かがわかる場合には，頻度が高いと考えられるほうに最適化されたコードを生成させることができる．また，CDR コーディングされる可能性が全くない場合には，CDR コーディングのオーバヘッドのない従来方式のリスト生成コードを生成すればよい．CDR-P を操作しない従来方式のコードは，CDR コーディングのコードと併用可能である．

3.2.3 項で述べたように，図 8 の partition/4 のようなプログラムでは，データの特性によって CDR コーディングの成功率が変化するために，従来のリスト処理方式に対する処理速度の比率が変動する．このため，一般的なプログラムにおける CDR コーディング方式を用いた処理系の処理速度を正確に予測することは容易ではない．しかしながら，図 12 には CDR コーディングが行われた場合と行われなかった場合の，リストの参照と生成の処理速度の最良値と最悪値が示されている．したがって，選択点と環境の操作や，リスト以外の単一化処理の処理時間を考慮する必要があるが，プログラムやデータの特性によって CDR コーディングの成功率が算出できれば，従来の

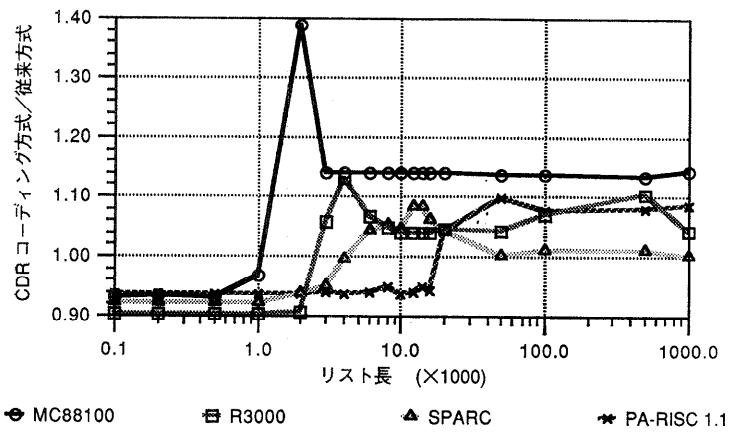


図 13 性 能 比
Fig. 13 CDR Coding method/Conventional method performance ratio.

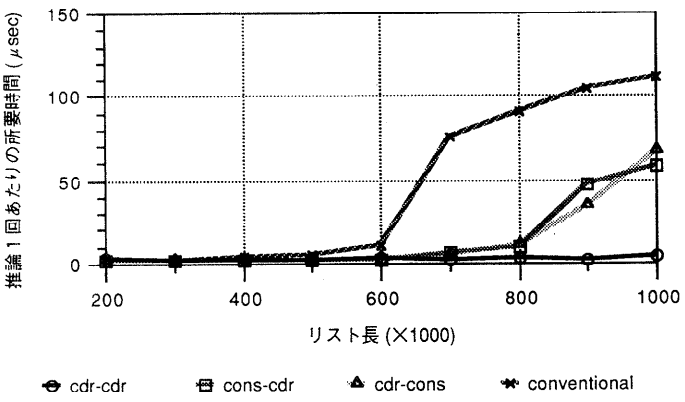


図 14 応答時間
Fig. 14 Response time.

処理方式に対する性能比の概略を求めることができる.

4.5 応答時間の改善

ワークステーションなどのパーソナルマシンでは, CPU 時間を少し多く消費しても, 実時間での応答時間が短いほうが利用者にとって利益が大きい. CDR コーディング方式を使用すると, メモリの使用量が減少するために塵集めの頻度が低下すると共に, メモリ参照が局所化されるためにページングの頻度が低下し, 実時間における応答時間が改善されるという利点がある. R 3000 を用い 16 M バイトの実メモリを実装したマシンで, 処理を行うリストの長さを変えて, append/3 の処理に要する経過時間を測定し, 推論1回あたりの所要時間を求め**図 14** に示した.

×で示される従来方式は, リスト長が 600,000 程度から, 入力あるいは出力のみが CDR コーディングされている□と△の系列は, リスト長が 800,000 程度から所要時間が大幅に増大し始めている. これらの所要時間の増大開始点では, 両者とも 10 M バイト弱のメモリを使用している. 一方, ○で示される CDR コーディング方式は, 図の右端の部分でもメモリの使用量は 8 Mバイト程度で, 所要時間はほぼ一定の値を保っている. 所要時間がメモリの使用量で 10 M バイト程度から増加しはじめるのは, OS や X ウィンドウサーバなどのプロセスが 6 Mバイト程度の実メモリを占有しているためと考えられる. この評価では, リスト長が長くなりメモリの使用量が大きくなると, 図 14 の右端に示されるように, CDR コーディング方式のほうが 40 倍ほど所要時間が短くなり, 応答時間が速くなるという結果が得られた.

5. まとめ

本論文では, 次のような特長を持つ PROLOG 処理系の動的 CDR コーディング方式を提案し, 性能評価を行うことによってその有効性を示した.

1) リストを従来の2倍のメモリ効率で表現できる.
2) 通常の単一化処理で実現される.
3) CDR コーディングの可能性を動的に検査する.
4) 従来の方式と同等の処理速度が得られる.
5) メモリの参照頻度を低減できる.

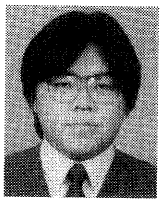
本方式は, 従来方式と比較してメモリの参照頻度が低く参照の局所性が高いために, 速度低下の要因となるキャッシュミスなどの影響を受けにくく, RISC や Superscalar などの今後の高速プロセッサに適した実現手法であると考えられる.

今後の課題としては, リスト操作に対して様々な特性を持つプログラムによる性能評価や, 操作されるリストのワーキングセットを考慮した性能評価を行いたいと考えている.

参考文献

1) 砥崎賢一ほか: PROLOG 処理系における CDR コーディング方式, 第 36 回情報処理学会全国大会論文集, pp. 801-802 (1988).
2) 砥崎賢一ほか: PROLOG処理系アーキテクチャの拡張と最適化方式の提案, *Proc. of the Logic Programming Conference '88*, pp. 151-160, ICOT (1988).
3) 砥崎賢一: PROLOG 処理系の CDR コーディング方式の性能評価, 情報処理学会記号処理研究会, SYM-62-3 (1991).
4) Dobry, T. P. et al.: Design Decisions Influencing the Microarchitecture for a Prolog Machine, *Proc. of the 17th Annual Microprogramming Workshop*, pp. 217-231, IEEE CS (1984).
5) Warren, D. H. D.: An Abstract Prolog Instruction Set, *SRI International, Technical Note 309* (1983).
6) Moon, D. A.: Architecture of the Symbolics 3600, *Proc. of the 12th Annual International Symposium on Computer Architecture*, pp. 76-83, IEEE CS (1984).
7) MC 88100 User's Manual, Motorola Inc. (1988).

(平成3年 9 月 2 日受付)
(平成4年11月12日採録)



碓崎　賢一（正会員）

昭和35年生．昭和60年大阪大学工学部造船工学科卒業．昭和62年同大学院基礎工学研究科情報処理分野博士前期課程修了．昭和63年同大学院博士後期課程退学．同年九州工業大学情報工学部助手．記号処理言語とその応用，オブジェクト指向言語とその応用，ソフトウェア開発支援環境，文書処理システムなどの研究に従事している．電子情報通信学会，人工知能学会各会員．