Common ESP処理系における最適化の一考察(1)

3Q-5

松浦 聡　田中 吉廣　実近 憲昭
㈱ AI言語研究所

1. はじめに

Common ESP (CESP) はICOTで開発されたESPを基に、汎用ワークステーション上で稼働するシステムを構築し、さらに機能を高めた言語である。言語体系としては、論理型とオブジェクト指向型の両パラダイムを融合した言語であり、AIプログラムの開発および、システム記述に適している。現在CESPは暫定版の開発を終了し、これに基づき、最終版の基礎となる基本仕様版の開発途中である。本論文はCESP基本仕様版における論理型言語部分の実行方式についての最適化とその性能評価について述べる。

2. CESP暫定版で採用した方式

前年度に開発したCESP暫定版では、WAMのエミュレータ方式で処理系を開発した[3] 。暫定版で採用したWAMの命令セットの構成を表1に示す。この内、次の最適化は実施済みである。

(a) if_then_else命令によるインデキシング
(jump_unless_list, jump_unless_unifiableなど)

(b) 複合命令の導入による最適化
(get_vart_vart_list, execute_deallocate など)

基本仕様版のWAMは、この暫定版のWAM命令セットを基にしている。

表1 CESP暫定版のWAM命令セット

種類	個数	代表的な命令
GET	18	get_vart_vart_list
UNIFY	13	unify_list
PUT	13	put_values_p
INDEXING	13	switch_on_term
CONTROL	7	proceed_with_deallocate
JUMP	23	jump_unless_list
CUT	6	cut_me_and_proceed
CALL	6	call
METHOD	14	call_method_immidiate
その他	3	allocate_exception_object

3. CESP基本仕様版の最適化方式

今回、CESPユーザプログラムでよく使用される記述に対応したWAMレベルの最適化を検討した。CESP基本仕様版では、コンパイラによりソースプログラムをWAMの中間言語まで翻訳し、その後、機械語を生成する方式で開発を進めている。我々は、このWAMレベルの実証評価を行うため、最適化を組み込んだWAMのエミュレータを作成し、最適化に関する性能評価を行った。次の項目について最適化を検討した。

(1) allocate, deallocate の省略
(2) if_then_else命令の強化
(3) 内部データベース用命令の追加

以下、これらの最適化の内容について述べる。

3.1 allocate, deallocate の省略

CESPで、再帰呼び出しのプログラムを作成した場合、そのWAMの実行では、Environment stack に同じデータが積まれる場合が多いが、再帰呼び出しのたびに同じデータをallocate, deallocate 命令で積み卸しを行うのは無駄である。従来方式では図1のプログラムを図2 (1) のようなWAMに変換していたが我々は図2 (2) のコードに変換している。

この方式は、ループ外にallocate, deallocate を出すことにより、1回のallocate, deallocate で実行が可能となる最適化である。

```
loop(0,X,X):-!;
loop(N,X,Z):-
    p(X,Y),
    !,
    loop(N-1,Y,Z);
```

図1 プログラム1 ループのプログラム

```
$loop   jump_unless_unifiable   A0, int!0, $loop_2
                :
        cut_me_and_proceed
$loop_1 trust_me_else_fail
$loop_2 allocate                 3
        get_variable_p           A0, Y0
                :
        execute_with_deallocate  $loop
```

図2 (1) プログラム1の変換例 (従来)

```
$loop   allocate                 3
$loop_0 jump_unless_unifiable    A0, int!0, $loop_2
                :
        cut_and_proceed_with_deallocate
$loop_1 trust_me_else_fail
$loop_2 get_variable_p           A0, Y0
                :
        execute                  $loop_0
```

図2 (2) プログラム1の変換例 (改良)

allocate, deallocate の省略が可能な条件として
(1) 節の数が2つである自己再帰的なプログラムである。
(2) インデキシングにより選択肢が積まれない。
(3) 決定的なプログラムである。
であるが、(3) の条件の判断は負担が大きいので我々は
(3')再帰の前に！がある。
を代わりの条件としている。

An optimizu method for Common ESP System(1)
Satoshi Matsuura, Yoshihiro Tanaka, Noriaki Sanechika
AI Language Research Institute, Ltd.

3.2 if_then_else命令の強化

大小比較を行う場合に片方が整数である場合は比較的多い（ＣＥＳＰシステムでは９０%）。そこで、我々は暫定版のif_then_else命令に整数と大小比較を行う命令（jump_unless_costant_less_than,etc）を追加した。

jump_unless_costant_less_than 命令の出る条件は

(1) jump_unless_less_than Ax,Ay,Lab が出る。

(2) jump_unless_less_than 命令において Ax または Ay が0～65535 の整数。

である。以下にjump_unless_costant_less_than のコードの出力例を示す。

```
abs(A, B) :-
    A > 0, !, A = B;
abs(A, B) :-
    B = -A;
```

図3　プログラム2　絶対値を求めるプログラム

```
$abs    put_integer             A2,int!0
        jump_unless_less_than   A2,A0,$abs_2
                :
        proceed
$abs_2          :
```

図4（1）　プログラム2の変換例（従来）

```
$abs    jump_unless_constant_less_than  A0,int!0,$abs_2
                :
        proceed
$abs_2          :
```

図4（2）　プログラム2の変換例（改良）

3.3 内部データベース用命令の追加

ＣＥＳＰには内部データベースをアクセスすることが多いにも拘らず、ＷＡＭにはデータベース用の命令がなかった。我々は内部データベース用のプログラムのために述語の引き数のユニフィケーションを一度に行う命令 get_argumentsを作成した。

get_arguments は引き数のベクタ{Arg0,...,Argn} と変数を含まないベクタ{t0,...,tn} のユニフィケーションを行う命令である。get_arguments は次の場合に出る。

(1) hash_on_value により、インデキシングが行える。

(2) 述語の頭部に変数が現れない。

従来方式と新方式のコードの出し方を示す。

```
a(data1_1,....,data1_n) :- !;
        :
a(datam_1,....,datam_n) :- !;
```

図5　プログラム3　データベースのプログラム

```
        try_me_else   n,$a_2
        get_constant  A0, data1_1
                :
        get_constant  An_1, data1_n
        cut_me_and_proceed
                :
$a_m    trust_me_else_fail
        get_constant  A0, datam_1
                :
        get_constant  An_1, datam_n

        cut_me_and_proceed
```

図6（1）　プログラム3の変換例（従来）

```
        try_me_else     n,$a_2
        get_arguments {data1_1,...,data1_n}
        cut_me_and_proceed
                :
$a_m    trust_me_else_fail
        get_arguments {datam_1,...,datam_n}

        cut_me_and_proceed
```

図6（2）　プログラム3の変換例（改良）

get-arguments 命令の採用により、get 系命令が引き数の数だけ出ていたのが１命令で済むようになった。また、get-arguments 命令の出る条件の中で(1) は必要ないので、さらに広範囲な適用が期待できる。

4．性能評価

上記のプログラム１～3に対して速度性能とコード量をＣＥＳＰ基本仕様版テストエミュレータで測定した。プログラム3ではm=256,n=4 とした。測定したマシンはＳＵＮ－３／６０（３ＭＩＰＳ）である。

表2　速度性能評価結果

	従来(LIPS)	改良(LIPS)	性能比(%)
プログラム1	5941	6486	109.2
プログラム2	8310	8996	108.3
プログラム3	4380	5639	128.7

表3　コード量性能評価結果

	従来(BYTE)	改良(BYTE)	性能比(%)
プログラム1	64	64	100.0
プログラム2	36	32	112.5
プログラム3	10024	3072	333.3

5．まとめと今後の課題

ＣＥＳＰ基本仕様版の処理系の最適化技術について述べた。最適化によりループの実行および、大小比較、データベースの検索が、それぞれ９%，８%，２９%の速度向上を得られた。コード量ではデータベースの検索で２３３%の向上が得られた。

今後、一般アプリケーションプログラムでの性能評価やＣＥＳＰ基本仕様版の機械語生成による性能測定も行っていく予定である。

また、研究課題としては、出現頻度の多いＷＡＭの組み合わせの検出による最適化の適用範囲拡大やモード宣言等のプラグマ導入時による最適化がある。

これらの研究により、ＣＥＳＰの高速実行を推進していきたい。

6．参考文献

[1] Warren,D.H.D.:An Abstruct Prolog Instruction Set, Technical Note 309,SRI International(1983)

[2] 立野ほか:PSI-II の機械命令セット評価、 情報処理学会計算機アーキテクチャ研究会(1989.1)

[3] 田中ほか :暫定版Common ESPシステムにおける実行方式 第３８回情報処理学会全国大会(1989.3)