段階的変換によるLispプログラムのベクトル化

1J－3

菅谷 正弘，安村 通晃
(株)日立製作所 中央研究所

1．はじめに

知識情報処理への要求が高まるにつれて、並列計算機あるいはスーパコンピュータによる非数値処理の高速化への期待も高まっている。従来、特にPrologに対して、非数値処理のベクトル化による高速化法の研究が行われてきたが[1,2]、Prologだけでは適用範囲が狭いことが一つの問題であった。

ここではベクトル型のスーパコンピュータにより非数値処理の実行を高速化する方法を確立するため、Lisp言語で記述されたプログラムを対象として取上げる。方式として段階的なプログラム変換により、具体的にはBoyer-Mooreの定理証明プログラムの高速化実験を行ったので報告する。

2．Boyer-Moreの定理証明プログラム

Boyerのプログラムは、Lispで記述された標準的なベンチマーク･プログラムであり[3]、主としてプログラムやLSI回路の論理の検証などに用いられている。

主な処理は、パターン･マッチと項の書き換えと、その結果に基づく真偽値の判定である。実行の内容は、書き換え規則を取り込む前準備的なsetupと、与えられた命題により引数を書き換え真偽判定をする実処理的なtestからなる。なお、setupの主要部分のプログラムを図1に示す。

3．段階的プログラム変換の方式

Lispで記述されたプログラムの自動ベクトル化に向けて、段階的プログラム変換によるベクトル化方式を用いた。変換は、次の段階的ステップからなる。

(1) LispからPascalへの書き換え
(2) Pascal上でのリストの配列化
(3) Pascal上での全ての関数の再帰呼び出しの除去
(4) PascalからFortranへの書き換え
(5) Fortran上でのgo to文などのdoループ化
(6) Fortran上での自動ベクトル化不能要因の除去

```
(defun add-lemma (term)
 (cond ((and (not (atom term))
             (eq (car term)
                 (quote equal))
             (not (atom (cadr term))))
        (setf (get (car (cadr term)) (quote lemmas))
              (cons term (get (car (cadr term))
                              (quote lemmas)))))
       (t (error "~%add-lemma did not like term: ~a" term))))

(defun add-lemma-lst (lst)
 (cond ((null lst) t)
       (t (add-lemma (car lst))
          (add-lemma-lst (cdr lst)))))
```

図1 Boyerのプログラム(setup部分)の主要部

4．ベクトル化の技法

ベクトル化のためのプログラム変換において用いた主な技法は、次のとおりである。

(1) リストの配列化

cdrコーディングを用いて、リストの要素を配列の要素に格納することにより変換する。

(2) 関数の再帰呼び出しの除去

スタックを用いて呼び出しの情報を退避させ、go to文を用いて制御を繰り返し構造へ変換する。

(3) go to文などによるループのdoループ化

繰り返し回数を解析し、doループへ変換する。

(4) 自動ベクトル化不能要因の除去

doループ内に存在する、関数呼び出しのインライン展開、文字型データの整数型データへの変換、長大ループの分割等を行う。

(5) 繰り返し構造の変換

自動ベクトル化不能要因となる図2の(1)に示した二重ループに対して、図2の(2)に示したような内外ループの入れ換えにより、最内側のループをベクトル化可能なdoループにする。

(6) 繰り返し構造の改良

配列要素間に逐次的な依存性があるために、単純なベクトル化では正しく動作しないようなループ対して、マスク･ベクトルを用いて同一のベクトル処理を繰り返すように改良することにより、ベクトル化可能なdoループにする。

Vectorization of a Lisp Program based on Stepwise Transformation
Masahiro SUGAYA, Michiaki YASUMURA
Central Research Laboratory, Hitachi, Ltd.

```
      DO 10 I = 1, VECTLENG
        ARRAY2(I) = VOID
        LIST = ARRAY1(I)
        DO
          IF (NULL(LIST)) EXIT
          IF (CAR(LIST) .EQ. QLEMMAS) THEN
            LIST = VOID
            ARRAY2(I) = CADR(LIST)
          ELSE
            LIST = CDDR(LIST)
          END IF
          REPEAT
10    CONTINUE
```

ベクトル化されない

(1) 変換前のプログラム

```
      REST = VECTLENG
      DO
        IF (REST.EQ.0) EXIT
        REST = 0
        DO 10 I = 1, VECTLENG
          LIST = ARRAY1(I)
          IF (.NOT. NULL(LIST)) THEN
            REST = REST + 1
            IF (CAR(LIST) .EQ. QLEMMAS) THEN
              ARRAY1(I) = VOID
              ARRAY2(I) = CADR(LIST)
            ELSE
              ARRAY1(I) = CDDR(LIST)
              ARRAY2(I) = VOID
            END IF
          END IF
10      CONTINUE
        REPEAT
```

ベクトル化される

(2) 変換後のプログラム

図2　繰り返し構造の変換による
プログラム変換

5. 実験結果

現時点では、Boyer のプログラム全体を Fortran までプログラム変換した後に、まず setup 部分からベクトル化を行った。しかし、setup の部分に限ってもその処理内容は非数値の代表的な処理を含み、今回の結果は十分に一般性があると言える。以下にその結果を述べる。

(1) 結果

リスト処理や関数の再帰呼び出し処理などの Lisp の中心的な処理を段階的プログラム変換によりベクトル化することができ、ベクトル化による高速化法は Lisp にも十分適用できた。

段階的プログラム変換の大部分の処理は自動的に行えることが判明し、Lisp プログラムの自動ベクトル化システム作成への可能性を確認できた。

(2) 性能評価

自動ベクトル化できない I/O 処理部分を除いた setup 部分のベクトル化率は、63 % であった。また、Fortran での実行については、ベクトル化を行う前のプログラムに対して、ベクトル化を行ったプログラムをベクトル実行した場合の実行速度は、1.44 倍に高速化できた。ベクトル化を行ったプログラムをスカラ実行した場合と比較すると、1.88 倍に高速化できた。これらを図 3 に示す。

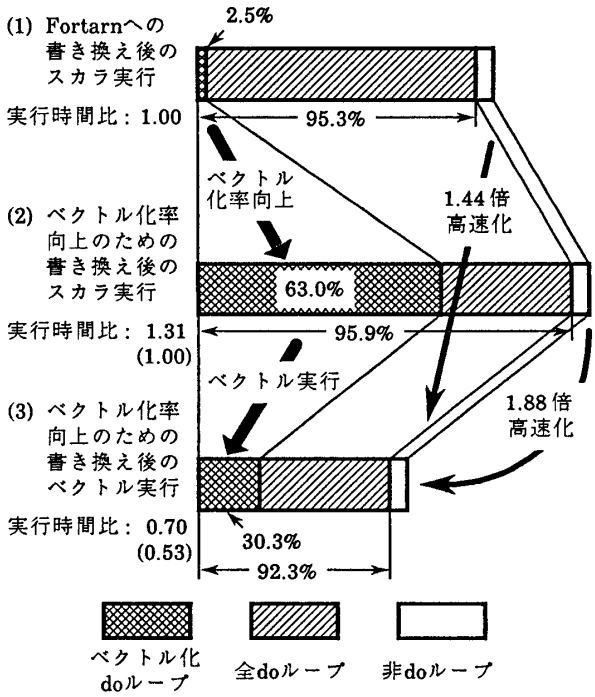

図3　Boyerのプログラム (setup 部分) の
ベクトル化率と実行性能比*

*：I/O 部を除いた S 810 での CPU 時間

6. おわりに

Boyer のプログラムを人手により段階的にプログラム変換してベクトル化し、ベクトル化による高速化法の有効性を確認した。自動化に向けては、さらに研究を進める必要がある。

参考文献

[1] 金田, 菅谷：プログラム変換にもとづくリストのベクトル処理方法とそのエイト・クウィーンへの適用：情報処理学会論文誌, Vol. 30, No. 7, pp.856-868, 1989

[2] 金田, 菅谷：OR並列実行のための論理型言語プログラムのベクトル化法：情報処理学会論文誌, Vol. 30, No. 4, pp.495-506, 1989

[3] Gabriel, R. P. : Performance and Evaluation of Lisp Systems : The MIT press, 1985.