Committed Choice による関数プログラムの並列実行

Parallel execution of functional programs by committed choice

田中 哲朗† 岩崎 英哉† 武市 正人†
Tetsurou TANAKA Hideya IWASAKI Masato TAKEICHI

†東京大学工学部
Faculty of Engineering, University of Tokyo

概 要

関数プログラムの並列実行系を実現するにあたって、関数型言語のプログラムを Committed Choice 型言語のプログラムに変換し、既存の処理系の上で実行する方式を考えた。これによって、並列度の解析ツールを利用して、関数プログラムの並列実行に関する実験を行うとともに、並列実行モデルとしての Committed Choice 型言語の妥当性を検証した。関数型言語として Gofer、Committed Choice 型言語として Fleng を用いた実験結果を提示する。

1 はじめに

関数型言語には並列性が内在しているため並列実行に適していると言われる．関数型言語の効率的な並列処理系を作るためには，プログラムのどの部分を並列に実行するか，また，並列化のためにどういう構文を導入するかなど決定すべきことが多い．これらの方針を決めるために，関数プログラムを既存の並列言語のプログラムに変換して実行することを試みた．

変換先の並列言語には，Committed Choice 型言語の 1 つである Fleng を選んだ．Fleng が動的なプロセス生成，動的なメモリ管理などの柔軟な性質を持つと同時に，並列の粒度が十分細かく，関数プログラムの並列実行の粒度とほぼ一致しているからである．

以下では，関数型言語の 1 つである Gofer から Fleng への変換と，並列度解析ツールを用いた並列構文の導入の評価について述べる．

1.1 関数型言語 Gofer

Gofer [2]は，Mark P. Jones が限定型 (qualified types) の研究のために作った言語で，おおまかな点では関数型言語の標準となりつつある Haskell のサブセットであり，型クラスに関するいくつかの点ではスーパーセットになっている．

1.2 Committed Choice 型言語 Fleng

Fleng [4]は，FGHC などの並列論理型言語とよく似ている．しかし，ゴールの実行の結果が真偽値を持たない点で論理型言語の枠におさまらないので，Committed Choice 型言語と呼ばれる．

Fleng プログラムは FGHC と違ってガード部を持たず，基本的には節の選択をヘッド部のマッチングだけでおこなうので，記述力が低くなっているが．それをマクロの導入によって補っている．quicksort プログラムの一部をマクロを使って書いた例を下にあげる．

```
partition(N,[],X,Y):-X=[],Y=[].
partition(N,[H|T],X,Y):-
    H>N ->
```

```
      X=[H|X1],partition(N,T,X1,Y)
    ;
      Y=[H|Y1],partition(N,T,X,Y1).
```

マクロを使うと非決定性がない述語は 1 つの節で書けるので，変換プログラムを作る上では有利である．

1.3 関数型言語から論理型言語への変換

関数型言語を論理型言語に埋め込む研究は，[5]や[3]などに述べられている．基本的なアイデアは，関数を評価した結果の値を返すために引数を 1 つ増やした述語に変換する点で共通している．

変換の際に問題になるのは，高階関数と遅延評価の実現法であるが，ここでは [3]で用いている方法と同等の方法をとった．

1.4 変換プログラム

Gofer から Fleng への変換プログラムの実現法としては，Gofer 処理系とも Fleng 処理系とも独立な変換プログラムを作る方法や，Gofer プログラムを読み込んで Fleng へ変換するプログラムを Fleng で書く方法などいくつかの方法が考えられる．

ここでは，Gofer 処理系のソースコードに手を加えて，Gofer プログラムを読み込んで型解析をすませた結果の中間コードを元に Fleng プログラムを出力できるようにした．この方法をとった結果，Gofer 処理系の型解析の結果を使って，多義性を持つ関数を容易に実現できた．

2 変換の概要

2.1 遅延評価

Gofer で書いた tarai のプログラムを素直に Fleng に変換すると次のようになる．

```
tarai x y z =
  if x>y then
    tarai (tarai (x-1) y z)
          (tarai (y-1) z x)
          (tarai (z-1) x y)
  else y
->
g_tarai(X,Y,Z,R):-
    (X>Y->
      X1 is X-1,g_tarai(X1,Y,Z,R1),
      Y1 is Y-1,g_tarai(Y1,Z,X,R2),
      Z1 is Z-1,g_tarai(Z1,X,Y,R3),
      g_tarai(R1,R2,R3,R)
    ; R=Y).
```

この変換は先行評価に対応していて，並列度は上がるものの不必要な計算がおこなわれる．値が必要になるまで評価をおこなわない遅延評価を実現するために，値を必要とする側が変数を長さ 1 のリストに具体化することとし，それを待って値を生成する側がリストの中身を具体化するという形の要求駆動型プログラムに変換する．前出の tarai プログラムを書き直すと次のようになる．

```
g_primGtInt(A1,A2,[R]):-A1=[X],A2=[Y],
    (X>Y->R=true;R=false).
g_primSubInt(A1,A2,[R]):-A1=[X],A2=[Y],R is X-Y.
g_tarai(A1,A2,A3,[R]):-
    g_primGtInt(A1,A2,[R1]),
    (R1==true->
      g_primSubInt(A1,[1],R2),g_tarai(R2,A2,A3,R3),
      g_primSubInt(A2,[1],R4),g_tarai(R4,A3,A1,R5),
      g_primSubInt(A3,[1],R6),g_tarai(R6,A1,A2,R7),
      g_tarai(R3,R5,R7,[R])
    ; [R]=A2).
```

2.2 高階関数

tarai のように大域的に定義された関数で，簡約化に必要な引数が揃っている場合の関数適用は，Fleng の対応する述語を直接呼び出す形に変換すればよい．しかし，一般には，関数型言語では関数を引数として渡したり，値として返すことができるので，それに対応する必要がある．

そこで，関数をクロージャとして表現し，述語 apply によってクロージャに引数を適用する方法をとった．クロージャの第 1 引数は簡約化に必要な残りの引数の数で，第 2 引数は確定した実引数のリストである．

```
tarai1 x = tarai (1+x)
flip f x y = f y x
soko = flip (tarai1 9) 0 5
->
g_tarai1(A1,[R]):-
    primPlusInt(A1,[1],R1),
    R=g_tarai(2,[R1]).
g_flip(F,X,Y,[R]):-apply(F,Y,R1),apply(R1,X,[R]).
g_soko([R]):-g_tarai1([9],R1),g_flip(R1,[0],[5],[R]).

apply(A,Y,[R]):-
    A=[{F,N,X}],
    (N>1 -> sub1(N,N1),R={F,N1,[Y|X]}
    ; apply1(F,X,Y,[R])).
apply1(g_tarai,[A1,A0],A2,R):-g_tarai(A0,A1,A2,R).
```

2.3 特殊形式

Gofer のデータ構成子 (constructor)，局所的な定義 (let, where)，場合分け構文 (case) などの特殊形式の変換例を示す．

```
-- constructor
test0 x = x : []
=>
g_test0(R0,[R1]):-R1 = ':'(R0, [nil]).
-- let, where
test1 n = n2 * (test1 n1) where
   n1 = n - 1
   n2 = n + 1
=>
g_test1(R0,[R1]):-
   g_primMinusInt(R0,[1],R2),
   g_primPlusInt(R0,[1],R5),
   g_test1(R2,R8),
   g_primMulInt(R5,R8,[R1]).
-- case
test2 0 = -1
test2 1 = -2
=>
g_test2(R0,[R1]):-
    [R2] = R0 ,
    (R2 == 0 ->
      R1 = -1
    ; (R2 == 1 -> R1 = -2)).
```

3 Gofer への並列性導入と Fleng 並列処理系による評価

並列性の評価には，Fleng 上の並列度解析ツール paf [6]を用いた．paf は無限個のプロセッサがあり，通信コストが一定であるというモデルを用いているので，子プロセスを作らずすぐに終了してしまうプロセスが多いと，見かけの並列度が大きくなるので，あまり参考にならない．そのため，CPU4 台構成の LUNA-88K 上の xfleng 処理系でプロセッサ台数を変えた場合の実行時間も提示することとした．

関数型言語に特別の構文を導入しなくても，組み込み関数の必須性によって並列に実行されるプログラムはあるが，ここでは並列構文の導入によってもたらされる並列性について扱う．

3.1 関数 spec による並列性

関数型言語に並列性を導入する方法の 1 つとして，関数f の適用と引数x の簡約化を並列におこなうspec f x = f x となる組み込み関数spec を用意する方法が考えられる．これは Fleng では次のように書ける．

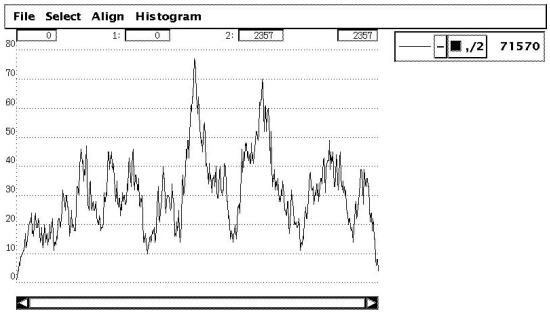

図 1　foldl 版 queens 4 の並列度

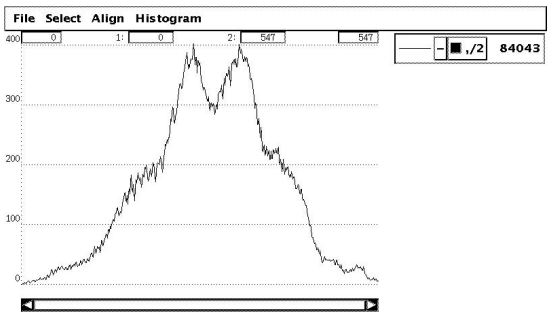

図 2　foldl_spec 版 queens 4 の並列度

表 1　foldl 版 queens 4 の実行時間

台数	実行時間 (mSec)	speed up
1	990	1
2	890	1.11
3	1110	0.89
4	1070	0.92

```
g_spec(A0,A1,[R]):-A1=[X],apply(A0,A1,[R]).
```

このspec を使って逐次版foldl を書き直すと次のようになる．

```
foldl_spec f z []       = z
foldl_spec f z (x:xs)   = spec (foldl_spec f) (f z x) xs
```

逐次版 foldl を使ったqueens をfoldl_spec を使うように書き直すと並列度が上がり，プロセッサ台数を増やした時の実行時間が短くなることが，paf による並列度表示 (図 1, 図 2)，と LUNA-88K による並列実行 (表 1, 表 2) の結果から観察される．

3.2 引数による並列性制御

前の節の例では，プログラム自体を書き換えているので，結果を使う側がリストを最後までたどらな

表 2　foldl_spec 版 queens 4 の実行時間

台数	実行時間 (mSec)	speed up
1	1190	1
2	700	1.7
3	550	2.16
4	480	2.58

いtake のような関数の場合は無駄な計算をおこなうことになる．そこで，queens のプログラムの引数として，実行を制御するための関数を渡す方法を試みた．

```
lengthS x = length (x spec)
takeS n x = take n (x id)
queens1 n = queensS n id
queens2 n = queensS n spec
queensS n spec_or_id = do_row 1 empty_board []
  where
    do_row row board solns =
      foldlS try_col solns [1..n] spec_or_id
    where
      try_col solns col =
        if (legal_move board row col) then
          if (row == n) then
            newboard : solns
          else
            do_row (1 + row) newboard solns
        else
          solns
      where
        newboard = update_board board row col

foldlS f z xs spec_or_id =
  foldlS_1 spec_or_id f z xs
foldlS_1 spec_or_id f z [] = z
foldlS_1 spec_or_id f z (x:xs) =
  spec_or_id (foldlS_1 spec_or_id f) (f z x) xs
```

上の定義で，queens はlength する時は並列実行，take されるときは逐次実行と必要に応じて動作が変わる (図 3, 図 4)．必須性解析をしてから，書換えを自動的におこなうことができれば，並列性を意識しないで書かれた多くの関数プログラムの並列動作に役立つものと考えられる．

4　おわりに

関数型言語 Gofer を Committed　Choice 型言語 Fleng に変換することによって，明示的に並列性を導入し，並列実行することができた．非決定性の導入などの拡張もこの方法で可能になるものと考えられる．

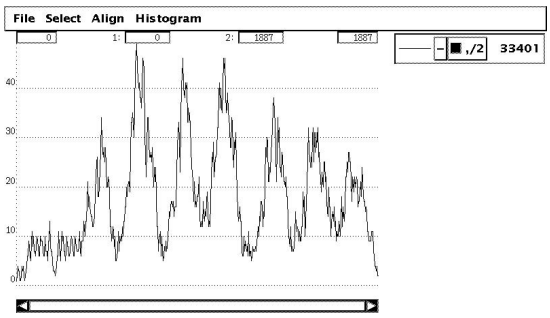

図 3　takeS 1 (queensS 4) の並列度

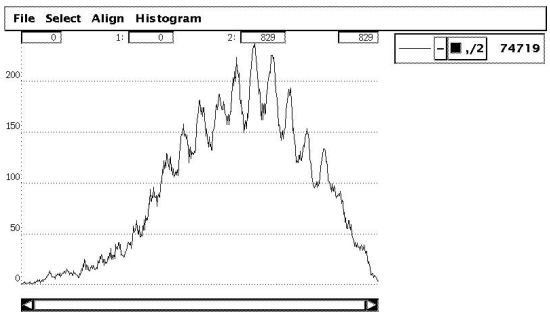

図 4　lengthS (queensS 4) の並列度

本研究を進めるにあたって，Fleng 処理系を使わせて頂き，また，さまざまな質問に答えて下さった東京大学電気工学科の田中英彦研究室の方々に心から感謝いたします．

参考文献

[1] Bird, R. and Wadler, P.(武市正人 訳): 関数プログラミング，近代科学社, 1991.

[2] Jones, M. P.: An Introduction to Gofer, University of Oxford, 1991.

[3] Levy, J. and Shapiro, E.: CFL-A Concurrent functional language embedded in a concurrent logic programming environment, Concurrent Prolog: Collected Papers. Vol. 2, E. Shapiro, Ed. MIT, Press, Cambridge, Mass., Chap. 35, pp. 442-469.

[4] Nillson M. and Tanaka H.: Fleng Prolog - The language which turns Supercomputers into Prolog Machines, In Wada, E.(Ed): Logic Programming'86, LNCS264,Springer-verlag(1989).

[5] 繁田良則, 赤間清, 宮本衛市: 関数型言語から論理型言語への変換について, 日本ソフトウェア科学会第 8 会大会論文集, pp. 281-284(1991).

[6] 白木長武，舘村純一，小池汎平，田中英彦：高並列プログラムのパーフォーマンス・デバッギング・ツール Paf, 情報処理学会第 8 回プログラミング - 言語・基礎・実践 - 研究会 SWoPP '92, 1992 年 8 月.