Committed-Choice 型言語 Fleng のインライン展開による粒度制御手法

荒木 拓也, 田中英彦
東京大学工学部

1 はじめに

Committed-Choice 型言語 Fleng は, 単一代入変数によって同期をとりながらすべてのゴール (Fleng における計算の単位) を並列に実行することにより、非定型的な問題においても すべての並列度を抽出して実行可能な言語である。しかし, 実行の粒度が非常に細かいため, オーバーヘッドが大きい. このオーバーヘッドを取り除くためには, 粒度を大きくする必要があるが, これは容易ではない.

粒度を大きくする手法としては, インライン展開と, 複数のゴールを 1 つのゴールに融合するゴール融合が考えられるが [1], 通常のインライン展開は, 同期条件が満たされている場合にのみ可能であるため, 適用可能な場合が非常に少ない. また, ゴール融合はプログラムの意味を変える可能性があるため, プログラム全体のデータフロー解析を行なう必要があり, 大規模なプログラムに適用することは困難である.

本稿では, Fleng プログラムをインライン展開できる形に変換することにより, 任意のゴールをインライン展開可能にする手法を提案する. この手法を適用することにより, 大規模なプログラムでも容易に粒度を大きくすることが可能になる.

2 Committed-Choice 型言語 Fleng

Fleng は論理型言語を祖先とする並列記号処理言語である. シンタックスは prolog のものと良く似ているが, バックトラックを行なわないという点で, セマンティクスは大きく異なる. Committed-Choice 型言語には, 他に GHC, KL1 などがあるが, Fleng は GHC からガードゴールを取り除いたものに相当する.

Fleng は,

- すべてのゴールを並列に実行する.
- 単一代入変数を用いたデータフロー同期をとる.

ことにより, プログラムの持つすべての並列性を抽出可能である.

例えば絶対値を求めるプログラムは以下のように記述される.

```
abs(A,R) :- greater(A,0,IsGt),abs1(IsGt,A,R).
abs1(true,A,R):- R = A.
abs1(false,A,R):- sub(0,A,R).
```

':-' の左側をヘッド部, 右側をボディー部と呼ぶ. abs というゴールをを呼び出すと, greater と abs1 がフォークされ, それぞれが並列に実行される. greater は A が 0 より大きいと IsGt を true に束縛し, そうでないと false に束縛する. abs1 は IsGt の値によって分岐し, true の場合は R = A を実行し, false の場合は sub(0,A,R) というゴールをフォークする.

ここで, abs1 の実行を開始するためには IsGt の値が必要だが, この値は greater の実行が終らないと決定されない. abs1 は実行を開始しようとしたときに IsGt の値が決まっていない場合, サスペンドする. そして IsGt の値が決まるとアクティベイトされる. このようにして, Fleng ではゴール同士の同期をとっている.

また, 算術演算は,

```
greater(#A,#B,R):- compute(>,A,B,R).
```

のように Fleng レベルで定義されている. ここで, '#' のついている変数は, 値が決まっていなければサスペンドするということを表す. compute は値が決まっていることを前提に, サスペンドせずに実行する.

3 インライン展開法

通常, インライン展開できるゴールは同期条件が満たされているものに限られる. 例えば, 先ほどの abs の中で呼ばれている greater や abs1 はインライン展開できない. これは, abs の第 1 引数である A が決定していなかった場合, greater や abs1 をサスペンドしなければ正しい値が得られないためである.

全てのゴールをインライン展開可能にするため, Fleng を拡張して, プログラムを以下のように変換する. greater を例にとると,

```
greater(A,B,R):-
    (isvar(A)-->
        suspend(greater(A,B,R),A)
    ;isvar(B)-->
        suspend(greater(A,B,R),B)
    ;compute(>,A,B,R)
    ).
```

An inlineing-based granularity control method of a committed-choice language Fleng
Takuya ARAKI, Hidehiko TANAKA
Faculty of Engineering,the University of Tokyo

のようにする．ここで，Cond --> Then ; Else は条件分岐を表す．この分岐は Non-blocking であり，逐次言語の条件分岐と同じ意味を持つ．ここでの Cond の部分 isvar は変数が束縛されているかどうかを調べるという意味である．また，suspend は第 1 引数に与えられたゴールをサスペンドさせる (第 2 引数にサスペンドの原因となる変数を与えているのは，サスペンドする際に，その変数にゴールをフックする必要があるためである)．

変換後のプログラムは，A または B が束縛されていない場合は greater はサスペンドし，A と B が束縛されている場合は compute(>,A,B,R) を実行するという意味になり，変換前のプログラムの意味と一致する．

このように Non-blocking の分岐と suspend という拡張を Fleng に行なうことにより，同期を行なう部分をヘッドからボディーに移すことができる．このように，ヘッド部で同期を行なわないコードに変換するとインライン展開が可能になる．abs の場合次のようになる．

```
abs(A,R) :-
    (isvar(A)-->
        suspend(greater(A,0,IsGt),A)
    ;
        compute(>,A,0,IsGt)
    ),
    abs1(IsGt,A,R).
```

この場合，A が束縛されている場合は greater のゴールフォークのオーバーヘッドが削減される．さらに abs1 もインライン展開し，整理すると以下のようになる．

```
abs(A,R) :-
  (isvar(A)-->
    suspend(greater(A,0,IsGt),A),
    suspend(abs1(IsGt,A,R),IsGt)
  ;
    compute(>,A,0,IsGt),
    (IsGt=true --> R = A; compute(-,0,A,R))
  ).
```

この場合，A が束縛されている方の分岐では，IsGt は compute によって束縛されていることが保証されるので，abs1 をそのまま展開することができる．また，A の値は束縛されていることが保証されているので，sub(0,A,R) もインライン展開することができ，最終的には A が決定している場合は，1 つのゴールで計算を実行することが可能になる．

この手法を用いると，再帰しているゴールをインライン展開することも可能であり，これはループアンローリングを行なうことに相当する．

4 議論

このようなインライン展開を行なった場合，必要となる変数が束縛されていた場合は十分な効果が得られるが，束縛されていなかった場合は効果が出ない．しかし，プログラムのサスペンド率はおおむねそれほど高くはないため，全体として十分な速度向上を達成することが可能であると考えられる．

また，ゴールに融合を行なった場合は変数が束縛されていない場合でも効率が向上する．また，ゴール融合ではインライン展開に比較するとコードサイズがそれほど増大しない．したがって，データ依存関係が単純で解析可能な場合はできるだけゴール融合を適用すべきであり，ゴール融合が適用できない場合にインライン展開を適用すべきであると考えられる．

5 実装法

実装法としては，インライン展開すべきゴールに対してだけ，上記のような変換を行なうのが，もっとも単純なやり方である．しかし，このように同期をボディー部で行なうように変換したプログラムは，コンパイルしたコードのイメージに近い．したがって，コンパイラを 2 パス構成にし，まず，同期部分のコードをボディー部に移動すし，さらにそのコードの上でインライン展開を行ない，その後通常のコンパイルを行なうという方法が考えられる．

このような構成をとることは，コンパイラの単純化にも役立つ．例えば，クローズインデキシングのような最適化は 1 パス目で実現可能である．

2 パス目のコンパイラは，同期部分がボディー部に記述されているので suspend 述語を除けば逐次言語のコンパイラと同様なものになる．

また，どのゴールをインライン展開するかは展開後のプログラムのコードサイズと性能に大きく影響する．理想的にはプロファイラを用いてより多く呼ばれるゴールを検出し，それをインライン展開するようにすべきであろう．

6 まとめ

Fleng において，同期部分をボディー部に記述することによってインライン展開を行なう手法を提案した．この手法は Fleng 以外の Committed-Choice 型言語や，データフロー同期を機能として持つ他の言語にも適用可能であると考えられる．

参考文献

[1] 荒木拓也，小池汎平，田中英彦．Committed-choice 型言語 Fleng における静的粒度制御手法．情報処理学会第 51 回全国大会，Vol. 6，No. 2P-8，pp. 101–102，September 1995.