ゴールの融合による Committed-Choice 型言語 Fleng の最適化

1T－3

荒木 拓也，　中田秀基，　小池汎平，　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並列度を最大限抽出して実行可能な言語である。しかし、逐次にしか実行できない部分においてもこの機構を用いて実行するので、オーバーヘッドが大きい。本稿ではプログラムの逐次部分を抽出し、1つのゴールに融合することによって、Fleng プログラムの最適化を行なう手法について述べる。

2　Fleng

Fleng のプログラムは以下のような定義節を並べたものである。

$$\mathrm{Head} :- \mathrm{Goal}_1,\mathrm{Goal}_2,...,\mathrm{Goal}_n.$$

:- の左側をヘッド部、右側をボディー部という。Fleng の計算単位をゴールと呼び、実行可能なゴールが与えられるとゴールと同じヘッド部を持つ定義節が選択される。これをヘッドユニフィケーションという。このゴールは選択された定義節のボディー部の各ゴールに展開される。これをリダクションという。展開された複数のゴールはそれぞれ並列にヘッドユニフィケーション、リダクションを行なう。

変数は単一代入であり、書き換えることはできない。変数は未定義か、値を持つかの2つの状態しか持たず、未定義の変数を参照しようとするゴールはサスペンドする。サスペンドしたゴールは変数がバインドされた時点でアクティベイトされる。Fleng ではこの性質を利用して同期を実現する。

例として絶対値を求めるプログラムをあげる。

```
absolute(A,R):-greater(A,0,B),goal1(B,A,R).
goal1(true,A,R):- R = A.
goal1(false,A,R):- sub(0,A,R).
greater(#A,#B,R):-compute(>,A,B,R).
sub(#A,#B,R):-compute(-,A,B,R).
```

ここでヘッド部の# の付いた変数は、バインドされるまで待つことを意味する。また、compute,= はシステムに組み込みの述語であり、サスペンドせずに実行する。

このように、Fleng ではヘッドユニフィケーションの部分で、同期と分岐を実現している。

3　ゴールの融合

上の例ではgreater の出力する結果B を参照して、goal1 が分岐する。したがって、greater とgoal1 の間には、

$$\mathtt{greater} \rightarrow \mathtt{B} \rightarrow \mathtt{goal1}$$

というデータ依存関係があることがわかる。

このデータ依存関係のため、greater とgoal1 は逐次にしか実行できない。このような場合、単純にgreater とgoal1 を別のプロセッサに割り当てると、不要なリモートメモリの参照や、サスペンド・アクティベイトが起こり、実行速度が低下する。

そこで、greater とgoal1 を1つのゴールに融合することによって、逐次化を行なうことを考える。融合後の absolute は次のようになる。

```
absolute(A,R):-greater_goal1(A,R).
greater_goal1(#A,R):-
        compute(>,A,0,B),
        ((B = true)-->
             R = A
        ;
             sub(0,A,R)
        ).
```

ここで、A --> B ; C という分岐を表す記法を導入した。if(A) then B else C という意味である。この分岐はヘッドユニフィケーションで行なわれる分岐とは異なり、同期を伴わない。

この逐次化により、

- ゴールをフォークする回数が減る。
- goal1 がサスペンドしなくなる
- greater とgoal1 の通信につかわれていた変数B は、ゴール内でしか使われないのでプロセッサ間の通信がなくなる

---

*A optimizing method of committed-choice language Fleng with goal fusion
Takuya ARAKI, Hidemoto NAKADA, Hanpei KOIKE, Hidehiko TANAKA,
Faculty of Engineering,the University of Tokyo

というメリットがある[1]。

このような変換が可能なのはデータ依存関係があるゴールのうち、前のゴールが実行されると必ず次のゴールが実行可能になるようなものに限られる。データフローグラフでいうと、グラフの途中から入力の枝が出ていないものである。

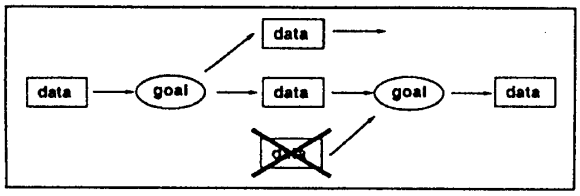

図 1: 変形可能な場合のデータフロー

上であげた例は 2 つのゴールを 1 つのゴールに融合するものだが、3 つ以上の場合もこの手法を繰り返し用いることにより融合可能である。

4 予備評価

本研究室では Fleng を高速実行するように設計された並列推論マシン PIE64[1] が稼働している。手で最適化、コンパイルしたものを PIE64 上で実行し、予備評価を行なった。対象プログラムは素数を求めるプログラム primes で、1000 までの素数を求めた。このプログラムは比較的並列度の低いものとして知られている。

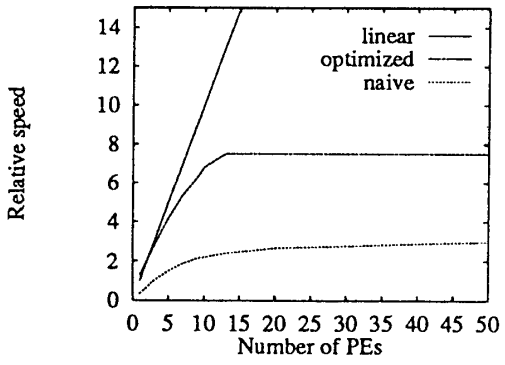

図 2: 予備評価

図の横軸はプロセッサ数、図の縦軸は相対速度である。相対速度は、アセンブラで逐次に書いたプログラムでの実行速度を 1 とした。primes のような並列度の低いプログラムにおいても台数効果がでていることから、最適化による並列度の低下が見られないことがわかる。また、アセンブラで書いたプログラムとの実行時間の比較でも、オーバーヘッドが少ないことがわかる。

[1] さらに、この場合は absolute からは greater_goal1 しか呼び出さないという特別な場合なので、greater_goal1 を absolute に置き換えることができる。これによりリダクション回数も減らすことができる。

5 検討

データ依存関係のあるゴールにおいても、部分的に並列実行が可能な場合がある。

```
foo:- a(Tmp),b(Tmp).
a(Tmp):- Tmp = true,goal_a1,goal_a2...
b(true):- goal_b1,goal_b2...
```

このようなプログラムにおいて b がサスペンドせずに実行できた場合、図 3のようにゴールのフォークを並列に実行できる可能性がある

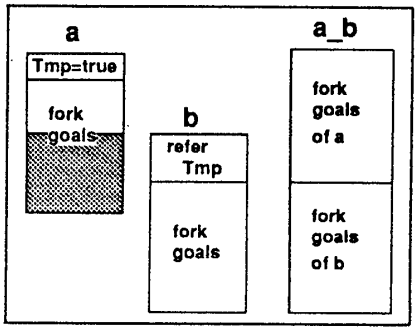

図 3: 部分的に並列実行が可能な場合

そのため、オーバーヘッドが減ることによるメリットと、並列性低下によるデメリットの両方を考慮して、このような逐次化を行なうかどうかを決定する必要がある。

図のようにサスペンドが起こらなかった場合で考えると、メリットは先に述べた通りで、デメリットは図の斜線部を並列に実行できないことであるから、斜線部の実行時間 < ゴールのフォーク + リモートメモリの参照のときにゴールの融合が可能である。

サスペンドが起こる場合は、サスペンド、アクティベイトがなくなることもゴールの融合によるメリットに付け加わる。しかし、サスペンドが起こるかどうかは実行される状況により変わるので、これを期待してゴールの融合を行なうと、実行速度が低下する可能性がある。したがって、サスペンドが起こらない場合を基準にして、ゴールの融合を行なうかどうかを決定する必要がある。

6 まとめ

ゴールの融合による Fleng プログラムの最適化手法を提案し、予備評価を行なった。また、並列度の低下についての検討を行なった。

参考文献

[1] 日高康雄, 小池汎平, 田中英彦. Pie64 の並列処理管理カーネルのアーキテクチャ. 情報処理学会論文誌, Vol. 33, No. 3, pp. 338–348, 1992.