5G−9

# データフロー解析に基づく Committed-Choice 型言語 Fleng の静的負荷分割

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、単一代入変数とサスペンド・アクティベイトの機構により、アルゴリズムの持つ並列性を最大限とりだすことのできる言語である。しかし、全てのゴール、変数を単純に分散して配置すると、データ依存関係にあり逐次にしか実行できないゴールも分散して配置され、本来必要のない通信コストがかかる。

データフロー解析を用いてプログラムから逐次部分を抽出することにより、並列度をさげることなく通信コストを低減する負荷分割が可能であり、これを行なうプログラムを実装した。本稿ではその手法について述べる。

## 2 研究の背景

### 2.1 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。

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

:- の左側をヘッド部、右側をボディー部という。Fleng の計算単位はゴールと呼ばれるものであり、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。ヘッドユニフィケーションされたゴールは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、リダクションが行なる。

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並列計算の同期を実現する。

*A static load partitioning method for Committed-Choice Language Fleng based on data flow analysis
Takuya ARAKI, Hidemoto NAKADA, Hanpei KOIKE, Hidehiko TANAKA,
Faculty of Engineering,the University of Tokyo

### 2.2 並列推論マシン PIE64

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 が稼働している。PIE64 は 64 台の IU (Inference Unit, 推論ユニット) によって構成され、各 IU がメモリ、プロセッサを持つ分散共有メモリ構成をとっている。また、それぞれの IU は回線交換方式のネットワークによって結合されていて、ネットワークに負荷情報を流すことにより負荷がもっとも少ない IU を自動的に選択することが可能な自動負荷分散機能を持つ。また、全ての IU の負荷が高い場合は、新たに生じた負荷をほかの IU に転送せず、過度の並列性は抑制されるので、並列度を上げることだけを考えればよい。

本研究は PIE64 上での Fleng の実行を対象にしている。

## 3 負荷分割戦略

### 3.1 負荷分割アノテーション

Fleng はゴールを生成することにより計算をすすめ、必要とするメモリは実行時に動的に確保する。よって、負荷分割を行ない得る場所として、ヒープメモリを確保する箇所とゴールを生成する箇所が考えられる。負荷分割戦略は、それぞれの箇所に次のようなアノテーションをつけることにより指定する。[1]

@[on(id)] ある変数が id で表される IU に存在することを示す。

@[to(id)] id で表される IU に変数、ゴールを転送することを示す。

@[any(id)] 負荷が最小の IU に変数、ゴールを転送することを示す。id は転送先の IU を表す。

@[local] ローカルの IU を選択することを示す。

処理系はこれらのアノテーションを解釈し、負荷分散を行なう。また、これらのアノテーションは次節に述べる方法でプログラムにより自動的に付加される。

### 3.2 アノテーションの付加方法

通信コストの低減は主にローカルメモリの参照率向上によって達成される。[1]よって、ローカルメモリの参照率が向上するような負荷分割を行なうことを考える。

データフロー解析を行なうと、ゴール間のデータ依存関係を知ることができる。データの依存関係があるゴール同士は並列に実行することは出来ないのでこれらのゴールを同じIUに置いても並列度は低下しない。よって、これらのゴールと依存関係を作っている変数とを同じIUに置くことによって、並列度を低下させる事なくローカルメモリの参照率を向上することができる。

データフロー解析の結果は一般に1本道ではなく、グラフ状になる。この場合は、グラフの同じ枝の上にあるゴールはデータ依存関係にあり逐次にしか実行できないが、異なる枝の上にあるゴールは並列に実行可能であるのでデータフローグラフを複数の枝に切り分け、それぞれの枝について上で述べた方法でアノテーションをつければよい。

今回実装したプログラムは、まずモード解析を行なう既存のプログラムの結果を利用してデータフロー解析を行なう。得られたデータフローグラフを最長の枝から順に複数の枝に分割し、それぞれの枝を構成するゴール、変数に同じidをもつアノテーションを付加する。

例をあげる。

```
foo(R):-
  bar(A,B),add(A,1,C),sub(B,1,D),mul(C,D,R).
bar(A,B):- A = 1, B = 1.
```

このプログラムのデータフローを図1に示す。このデータフローの場合、図に示すように2つの枝に切り分けることができる。この2つのグループはそれぞれ並列に実行可能であるが、グループの中のゴールは逐次にしか実行できない。よって、それぞれのグループを別のIUに置くような負荷分割が行なわれる。

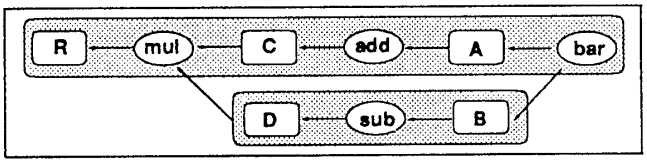


図1: データフロー

このプログラムにアノテーションをつけた例を次に示す。変数Rはヘッド部に現れていて、すでに置かれている場所が決まっているので@[on(1)]をつける。ゴールbar, add, mulと、変数A,Cには@[to(1)]をつけ、Rと同じ所に置く。また、Bに@[any(2)]を、subとDに@[to(2)]をつけ、ゴールsubと変数B,Dを1以外の同じIUに置く。

```
foo(R@[on(1)]):-
  bar(A@[to(1)],B@[any(2)])@[to(1)],
  add(A,1,C@[to(1)])@[to(1)],
  sub(B,1,D@[to(2)])@[to(2)],
  mul(C,D,R)@[to(1)].
bar(A,B):- A = 1, B = 1.
```

また、このときのメモリ参照の様子を図2に示す。ただし、アノテーションがついていない時は、変数はローカルに、ゴールは負荷最小のIUに転送するという単純な戦略をとるものとする。

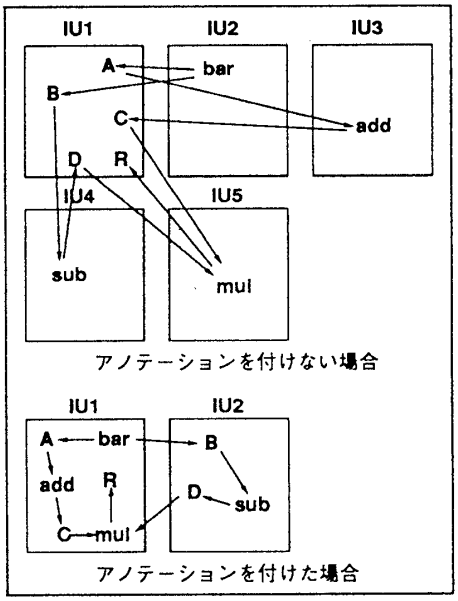


図2: メモリ参照の様子

この図のように、アノテーションを付加することによってローカルメモリ参照率が向上することが分かる。

## 4 おわりに

データフロー解析に基づいてFlengプログラムの負荷分割を行うことにより、ローカルメモリの参照率が向上し、通信コストが低減できることを示した。

現在、実行速度の改善を定量的に評価するために、これらのアノテーションを解釈する評価系を作成中である。

## 参考文献

[1] 日高康雄, 小池汎平, 舘村純一, 田中英彦. 実行プロファイルに基づくコミッティッドチョイス型言語 の静的負荷分散. 情報処理学会論文誌, Vol. 32, No. 7, pp. 807–816, 1991.

[1]通信コストにはリモートメモリの参照の他にゴールの転送もあるが、ゴールの転送時間は実行速度に影響しにくい。ゴールの転送は結果を待たないため、その間も他の処理ができ、レイテンシが隠れるためである。