PIE64 における Committed-Choice 型言語 fleng の動的粒度制御のためのコンパイル手法

1T－2

中田秀基, 小池汎平, 田中英彦
{nakada,koike,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学部

1 はじめに

並列実行の要点の一つに粒度の制御がある。細粒度言語の効率的実行には実行単位をまとめあげる操作が不可欠である。最適な実行粒度はその時点でのプログラムの並列性に依存するため、静的には定まらない。

我々は、Committed-choice 型言語 fleng に対してことなる粒度に調整した複数のコードを実行時の状態に応じて切替えることによって常に適切な粒度で実行する手法を提案し、並列推論エンジン PIE64 に実装を行なっている [中田 94]。我々はプログラムの粒度を調整するために、プログラム変換を用いる。プログラム変換は、潜在的にデッドロックの可能性を秘めており、安全にプログラム変換を行なうには、静的な解析を行なう必要がある。本稿ではこのプログラム変換のための静的解析手法について延べる。

2 fleng と PIE64

fleng は、並列論理型言語を祖先とする Committed-choice 型言語の一つである。GHC、KL1 と同族の言語であるが、ガードゴールをもたずヘッド部のパターンのみでガードを構成する点に特徴がある。

PIE64 は fleng 向けに開発された計算機である。各要素プロセッサは、計算、並列管理、通信 / 同期にそれぞれ特化した 3 プロセッサを持っているため、計算に負担を掛けずに、実行時の制御が行なえる。また、各要素プロセッサは、3 段のクロスバ・スイッチで接続されており、通信プロセッサが出力する負荷情報をスイッチが伝搬することにより、ある時点で最小の負荷値を持つ要素プロセッサとその負荷値をすべての要素プロセッサが知ることができる。負荷が均等に配分されると仮定すれば、最小負荷の要素プロセッサの負荷から、PIE64 全体の負荷が推定できる。実行時カーネルはこの機能によって得られた負荷に応じて、ことなる粒度に調整されたコードを切替えることで、実行粒度の制御を行なう。

3 粒度の調整

fleng の実行単位は、述語と呼ばれるものである。述語の実行は、1. ヘッド部のパターンマッチによる同期、2. 変数への値の束縛、3. ボディーゴールの作成、の 3 つからなる。通信という観点から見ると、2. の変数への束縛が送信、1. のパターンマッチが受信である。

fleng で実行粒を調整するには、実行単位である述語を融合すれば良い。具体的には、プログラムを展開することによって、ひとつの述語の実行長を長くするのである。ここで問題になるのは、1. が同期であるということである。安易に同期を融合すると、デッドロックの可能性を生じる。このため、融合する述語の解析を行ないデッドロックが生じないことを保証しなければならない。

4 静的解析手法

プログラムの意味を [UM89] に従って、可能な入出力の履歴の集合で定義する。可能なすべての入出力の履歴を与えるものとして、以下のようにプログラムの動作を定める。

4.1 プログラムの動作

プログラムの動作を下のように表現する。

- ガードグループ (ガードの集合)
- 束縛
- ガードグループと束縛の依存関係

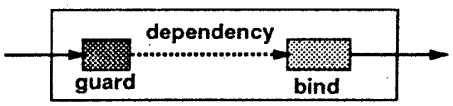

図 1: プログラムの動作

例えば、`test(a, B):- B = b.` の動作は下のように表せる。

```
guardGroup(0) = { guard([$head(1)], $sym(a))}.
bind(0) = unify([$head(2)], $sym(b)).
depend(bind(0)) = {guardGroup(0)}.
```

ここで定義されたプログラムの動作は、ガードと束縛の間に半順序を定める。この半順序を満たすすべての履歴が、プログラムの意味となる。

4.2 意味の合成

親ゴールの意味はすべてのサブゴールの意味と親ゴールのガード、束縛から合成される。これは、[UM89] でいう同期つきマージに相当する操作である。合成は以下の手順で行なう。

- 各サブゴールのガード、束縛の変数名を変換する。

A compile method for dynamic granularity control of Committed-Choice Language fleng exection
Hidemoto NAKADA, Hanpei KOIKE,
Hidehiko TANAKA
Faculty of Engineering, University of Tokyo

- 各サブゴールのガードに関して、ガードを満足させる束縛の組があるかをチェックする。なければ、新たに親ゴールのガードとして登録する。
- 親ゴールの外から見える束縛を選ぶ
- その束縛を実現するために必要なガードを選択して、依存関係を再構築する。

以下に例をあげて上記の手順をしめす。

```
test(a,B,C,E):-test1(B,D),test2(D,C),E=e.
test1(b, D):-D=d.
test2(d, C):-C=c.
```

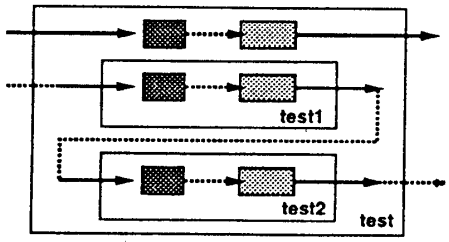

図 2: test プログラムの動作 (1)

まず、test1、test2 の動作の変数名を置き換えたものは以下のようになる。

```
guardGroup(1,0)={guard([$body(1,1)],$sym(b))}.
bind(1,0)       = unify([$body(1,2)],$sym(d)).
depend(bind(1,0)) = {guardGroup(1,0)}.

guardGroup(2,0)={guard([$body(2,1)],$sym(d))}.
bind(2,0)       = unify([$body(2,2)],$sym(c)).
depend(bind(2,0)) = {guardGroup(2,0)}.
```

さらに、親ゴールのガードと束縛を求める。

```
guardGroup(0,0) = {guard([$head(1)], $sym(a))}.
bind(0,0) = unify([$head(4)], $sym(e)).
bind(0,1) = unify([$head(2)], [$body(1,1)]).
bind(0,2) = unify([$head(3)], [$body(2,2)]).
bind(0,3) = unify([$body(1,2)], [$body(2,1)]).
depend(bind(0,*)) = {guardGroup(0,0)}.
```

次に、各ガードグループを満足させる束縛の組を求める。guardGroup(1,0) を満足する束縛の組は存在しない。guardGroup(2,0) は、bind(1,0), bind(0,3) によって満足される。前者を外に見せるガードとし、後者は内部的なガードとして外からは隠蔽する。前者をあらたにguardGroup(1) とする。

次に、外から見える束縛の組を求める。head(n) からたどれる束縛は、下の二つである。

```
bind(0,0) = unify([$head(4)], $sym(e)).
bind(0,2),bind(2,0)=unify([$head(3)],$sym(d)).
```

これらを親ゴールの外部へ提供する束縛としてそれぞれ、bind(0)、bind(1) とする。

最後に、束縛とガードの依存を調べる。bind(0) は guardGroup(0) に依存する。bind(2,0) は guardGroup(2,0) に依存し、guardGroup(2,0) は、bind(1,0), bind(0,3) によって満足される。bind(1,0) は、guardGroup(1) に依存する。従って新たに得られる親ゴールの動作は下のようになる。

```
guardGroup(0) = { guard([$head(1)], $sym(a))}.
guardGroup(1) = { guard([$head(2)], $sym(b))}.
bind(0) = unify([$head(4)], $sym(e)).
bind(1) = unify([$head(3)], $sym(d)).
depend(bind(0)) = {guardGroup(0)}.
depend(bind(1)) = {guardGroup(0),guardGroup(1)}.
```

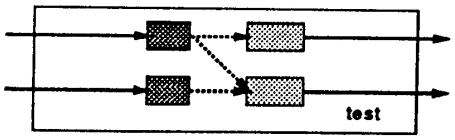

図 3: test プログラムの動作 (2)

4.3 完全展開可能性の検出

ある述語から下のすべてのサブ述語がプログラムの展開によって完全に直列化できるとき、その述語は完全展開可能であるという。完全展開が可能になるのは、以下の条件が成り立つ時である。

- サブ述語がすべて完全展開可能である
- サブ述語間に依存関係のループが存在しない
- すべての束縛が依存するガードグループの集合が同一である
- または、すべての束縛が依存するガードグループが少なくとも一つある

下のプログラムは、依存関係のループがあるため完全展開はできない。

```
test(C):-test1(A,B,C),test2(B,A).
test1(B,A,C):-test1s(B, C),A=a.
test1s(b,C):-C=c. test2(a, B):-B=b.
```

5 おわりに

fleng の粒度調整のための静的解析手法について述べた。現在、プログラム変換による粒度調整の実装を行なっている。今後、動的粒度制御に関して詳細に評価を行なっていく予定である。また、他の粒度調整法の導入についても検討中である。

参考文献

[UM89] Kazunori Ueda and Masaki Murakami. Formal Semantics of Flat GHC. 平成元年 電気・情報関連学会連合大会, 1989.

[中田 94] 中田秀基, 小池汎平, 田中英彦. fleng の動的粒度制御のための静的解析手法. 情報処理学会研究報告 プログラミング－言語・基礎・実践－, Vol. 94, No. 18, pp. 1–8, july. 1994.