

## 4Q-6

## FLENG コンパイラの最適化処理

下山 健, 島田 健太郎, 小池 汎平, 田中英彦

{ken,shimada,koike,tanaka}@mt1.t.u-tokyo.ac.jp

東京大学 工学部\*

## 1 はじめに

我々は現在、並列推論エンジン PIE64 [1] の開発を進めている。PIE64 上で大規模な知識処理を行なうための言語として、Committed Choice 型言語 FLENG、およびその上位言語 FLENG++[4] を採用している。PIE64 には、64 台の推論ユニット (Inference Unit: IU) があり、各 IU には、通信処理を受け持つ NIP (Network Interface Processor) と、FLENG の実行を直接受け持つ推論プロセッサ UNIREDDII [3] と、全体の管理を行なう汎用プロセッサ SPARC がある。UNIREDDII は、我々が先に概要設計をした UNIREDD を改良したものである。FLENG や FLENG++ を実行するには、これらのプログラムを、UNIREDDII のコードに落とすコンパイラが必要となる。PIE64 の性能を十分に引き出すためにはこのコンパイラの最適化処理は非常に重要である。本稿では、FLENG をコンパイルする際に考えられる最適化の一部と、UNIREDDII のアーキテクチャとして、コンパイラに想定している最適化について、その概要を述べる。

## 2 マクロの高速化

FLENG にはガードがないため、条件分岐を行なう時はヘッドのマッチング規則の機構を利用しなくてはならない。[4] このことを簡単に実現するために FLENG には 2 種類のマクロが用意されている。一方は、条件部を並列に調べる GHC ライクな Guarded-Command、もう一方は、条件部を逐次に調べる Prolog ライクな If-Then-Else と考えることができる。両者は、図 1 のように記述され、プリプロセッサによって展開される。これらは、FLENG の記述力や可読性を高めるために導入されているものである。しかし、プリプロセッサによって展開する場合以下のような部分に無駄があると考えられる。

- 条件部の論理値を各ゴール間で共有しているため条件判断のため複数のゴールが生成される。
- 条件分岐をヘッドのマッチング規則に委ねている。

このマクロを最適化のための情報としても利用することを考える。条件部には限られた比較演算等しか記述できないので、条件部のどれかが成立するような状態になってから、はじめて定義節をコミットするようにコードを生成する。こうすることにより、プリプロセッサで展開したものと意味は同等で余分なゴールを生成しないコードになる。ただし、このためには、マ

\*The Optimization on FLENG Compiler

## •GHC-like guarded-command

## 展開前

```
a(B,C):-B > 1 | b(B);
          C > 1 | c(C).
```

## 展開後

```
a(A,B):-gt(A,1,C),gt(B,1,D),a0(D,C,B,A).
a0(false,false,A,B).
a0(true,A,B,C):-c(B).
a0(A,true,B,C):-b(C).
```

## •Prolog-like If-Then-Else

## 展開前

```
d(B,C):-B > 1 -> b(B);
          C > 1 -> c(C).
```

## 展開後

```
d(A,B):-gt(A,1,C),d0(C,B,A).
d0(true,A,B):-b(B).
d0(false,A,B):-gt(A,1,C),d1(C,A).
d1(true,A):-c(A).
d1(false,A).
```

図 1: マクロとプリプロセッサによる展開

```
a(B,C):-B > 1 -> b(B);
          C > 1 -> c(C)), d(B,C).
```

```
a(B,C):-a0(B,C),d(B,C).
a0(B,C):-B > 1 -> b(B);
          C > 1 -> c(C).
```

最適化される部分

図 2:

クロと同レベルのボディゴールによって、具体化される変数が条件部になく条件である。そこで、マクロと同レベルのボディゴールが存在する場合、これを 2 つの定義節に分離して前述の最適化が適用できるように変形する。また、条件部がネスト構造になっていた場合、内部を一つのゴールとして展開する。(図 2) 条件部に and や or があつた場合等、マクロ内に逐次に解釈しなくてはならないところと、並列に解釈しなくてはならないところが混在する場合がありますので、条件判断には注意が必要である。

## 3 UNIREDDII 上の最適化

PIE64 上では、FLENG や FLENG++ は、UNIREDDII のコードとして実行される。FLENG の最適化には、前述のようにターゲットのマシンのアーキテクチャに依存しないものと依存するものに分けられるが、以下では UNIREDDII のアーキテクチャに依存するものを説明する。

### 3.1 頭部単一化が決定的な定義節のサスペンドの高速化

FLENG では、単一化可能な定義節が複数ある場合、その中から最初に頭部単一化に成功したものを非決定的にコミットしなくてはならない。これらをナイーブにコンパイルする場合、各定義節を独立にコンパイルしてつなげ、順番に調べて、全てコミットされなかった場合、サスペンド処理やフェイル処理を行なうようにコードが生成される。しかし、頭部単一化が決定的に行われるような定義節群の場合、サスペンドを決定するのに全ての定義節を調べる必要がない場合がある。たとえば、図4の `append/3` の場合、第一引数が未束縛のとき、残りの定義節を調べるまでもなくサスペンドすることは明らかである。したがって、この場合、すぐサスペンド処理を行なうのが好ましい。一方、`merge/3` のように頭部単一化が非決定的の場合、第一引数が未束縛でもコミットされる可能性はある。したがって、このような(数少ない)場合はナイーブにコンパイルする。

UNIREDI では、Passive Unification の命令として Dereference 系の命令が用意されている。一般に、Dereference をして、タグチェックをする場合、考えられる結果は3つある。

- タグが一致した場合 (結果1: コミット)
- タグが UNDEF だった場合 (結果2: サスペンド)
- タグがそれ以外だった場合 (結果3: フェイル)

前述のようにコミットが決定的に行なわれるような定義節のときは、それぞれの場合、コミット処理、サスペンド処理、次の定義節あるいはフェイル処理へ分岐できる。そこで、UNIREDI II では、Dereference をしてタグチェックをする命令では、その結果によって2通りに分岐できるようになっている。しかし、UNIREDI II の命令セットは1命令1ワードの原則があるので、双方の分岐のオフセットを十分に取ることは困難である。したがって、以下のように3種類のモードを設定してコンパイラが選択できるようになっている。

1. 結果2の分岐先をレジスタ、結果3を命令中に指定
2. 結果2,3の分岐先を共通にして命令中に指定
3. 結果2,3の分岐先を共通にしてレジスタで指定

ここで、結果2の分岐先をレジスタ指定にしたのは、このエントリを一連の定義節内で共通にできるからである。そのためには、各定義節でサスペンドを起こす原因となった変数を共通のレジスタに置かなくてはならない。最終的には、コンパイラは分岐先の距離、最適化によってこの3つのモードを使い分けることになる。

## 4 おわりに

以上で、FLENG コンパイラの最適化の一部について述べた。現在、このコンパイラは SUN4(SPARC)[6] と PIE64 (UNIREDI I & SPARC) をターゲットにして SUN4 の上で開発している。このコンパイラには本稿で概説した最適化技法の他にも、彩色法によるレジスタ割り当ての最適化[5]などを FL-ENG や UNIREDI II に応用して導入する予定である。

なお、本研究は文部省特別推進研究 No.62065002 による。

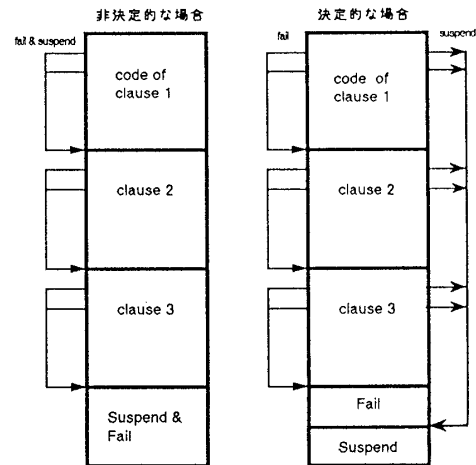


図3: 各定義節の関係

```
append/3
append([H|T], X, Z) :- append(T, X, Y), Z = [H|Y].
append([], X, Y) :- X=Y.
```

```
merge/3
merge([H|T], S2, S) :- S = [H|S3], merge(T, S2, S3).
merge(S1, [H|T], S) :- S = [H|S3], merge(S1, T, S3).
merge([], S2, S) :- S=S2.
merge(S1, [], S) :- S=S1.
```

図4: プログラム例

## 参考文献

- [1] Koike, H. and Tanaka, H.: "Multi-Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64" Proc. of Fifth Generation Computer Systems, Tokyo, Japan, November 1988.
- [2] Warren, D.H.D.: "An Abstract Prolog Instruction Set" Technical Note 309, Artificial Intelligence Center, SRI, 1983.
- [3] 島田, 下山, 清水, 小池, 田中: "推論プロセッサ UNIREDI II のアーキテクチャ" 情報処理学会計算機アーキテクチャ研究会 77-2, 1989年7月
- [4] 中村, 小中, 田中: "並列論理型言語 FL-ENG に基づいたオブジェクト指向言語 FL-ENG++" 日本ソフトウェア科学会, オブジェクト指向計算に関するワークショップ WOOC '89, 1989.
- [5] G.J. Chaitin Chow and John Hennessy: "Register Allocation by Priority-based Coloring" ACM SIGPLAN Notices, vol.19, No.6, June 1984, pp.222-232
- [6] 下山, 島田, 小池, 田中: "FL-ENG コンパイラとその抽象化コード" 情報処理学会第38回全国大会 6Q-3, 1989年3月