並列論理型言語 Fleng におけるプリミティブなオーバーヘッド解析

4N−3

馬場恒彦，荒木拓也，坂井修一，田中英彦
{tbaba,araki,sakai,tanaka}@mtl.t.u-tokyo.ac.jp
東京大学工学系研究科*

1 はじめに

並列論理型言語 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，非定型的な問題に対しても並列度を最大限抽出し実行可能な言語である．その一方で，この機構を実現する為のデータフロー同期によって，オーバーヘッドが引き起こされるという問題が生じる．

処理系に起因するオーバーヘッドに対する解析 [1] を行なったが，一方で Fleng の言語的に特徴に起因するプリミティブなオーバーヘッドに対する解析も必要である．

そこで，本研究ではその解析を以下のように行なった．まず，コンパイルしたソースに対し手動により最適化を行ない，コンパイラの成熟度に起因するオーバーヘッドの影響について解析を行なった．さらに，その結果得られたソースを解析することによりプリミティブなオーバーヘッドに関する考察を行なった．本稿では，その結果について述べる．

2 並列論理型言語 Fleng

並列論理型言語 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．

$$Head :- Goal_1, Goal_2, \ldots, Goal_n$$

:- の左側をヘッド部，右側をボディ部という．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 (ヘッドユニフィケーション)．選択されたゴールはボディ部の各ゴールに展開される (リダクション)．展開された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われる．

変数は単一変数であり，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，未定義な変数を参照しようとするとサスペンドする．サスペンドしたゴールは未定義な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している．

*Analysis of Primitive Overheads in Committed-Choice Language Fleng
Tsunehiko BABA, Takuya ARAKI, Shuichi SAKAI, Hidehiko TANAKA
University of Tokyo, Graduate School of Engineering,
7-3-1 Hongou, Bunkyo-ku, Tokyo 113, Japan

3 オーバーヘッド解析手法

Fleng ではゴール融合 [2] や強制インライン展開 [3] 等の最適化手法が提案・実装されているが，コンパイラ細部で改善の余地があり，削減可能なオーバーヘッドが存在する．

そこで，コンパイラの成熟度に起因するオーバーヘッドと Fleng の言語特徴に起因するプリミティブなオーバーヘッドとに切り分ける．それぞれに対し，以下のような手法により現在のコンパイラで得られるソースの解析を行なった．

1. コンパイラの成熟度による差異を極力除くためソースレベルでの手による最適化 (言語レベルでの最適化)．
2. 上述の方法で削除できなかったオーバーヘッドに対する逐次 C との比較による解析．

3.1 言語レベルでの最適化

3.1.1 Fleng レベルでの最適化

前述したゴール融合等の最適化手法は Fleng-to-Fleng のコンパイラとして実装されている．このコンパイラによって生成された Fleng プログラムにおけるオーバーヘッドについて解析を行なった．その結果，以下のような冗長なコードが生じており，その削減を行なった．

- 値の確定している変数に対する変数束縛チェック
- 分岐方向が一意に定まる場合の分岐条件チェック
- 同一分岐条件を個別に別途チェック

3.1.2 C レベルでの最適化

次に Fleng を変換して得られた C 言語レベルでの最適化を行なった．前節で述べた Fleng 言語レベルで手による最適化を行なった後の Fleng プログラムには以下のようなオーバーヘッドを持つ部分が見られた．

- 算術演算に関するオーバーヘッド
  Fleng では算術演算を行なう際にタグの着脱処理が必要である．そのため，算術演算が連続する場合においては一度付けたタグを再度外して演算することになる．複数の演算をまとめ，一度にタグの着脱を行なうことでこのオーバーヘッドを削減した．
  また，<, >, = といった比較を用いた分岐では，演算を行なった結果を真偽値 (true, false) で返し，再度その真偽判定を行なっている．そのため，比較を二度行なうオーバーヘッドが生じる．真偽値を介さず直接分岐することでこれを削減した．

- 確定している変数に対するチェック
  定数などのプログラム中で確定している変数に対して, 不要な参照外しを行なっており, 余計なオーバーヘッドが生じている.

以上, プログラム中に含まれるこれらの各部分を中心にそれぞれのソースに対して, ハンド・コンパイルを行ない, オーバーヘッドの削減を行なった.

3.1.3 結果

まず, コード量 (word 数) をまとめると表 1に示される結果となる. 表において, C 言語の word 数は cpp を用い, マクロ展開した word 数である. Fleng ソースにおいて約 20 〜 33% のコード量を, また Fleng から変換される C ソースでは, 約 7 〜 14% のコード量を削減することができた.

実行速度の測定結果のうち queen に関して表 2に示す. オーバーヘッドの削減により, 平均で約 4% の速度向上が得られることが示された.

表 1: 最適化によるコード量比較 (word 数)

	Fleng(*1)	Fleng(*2)	C(*1)	C(*2)
queen	3386	2512	43375	39293
gauss	17939	12084	220527	190610
primes	4180	3358	52525	49025

*1: コンパイラによるコンパイル後, *2: 手動による最適化後

表 2: Fleng での最適化による queen(n=11) の実行時間

スレッド数	1	2	4
最適化前	5.26 (s)	3.72 (s)	2.71 (s)
最適化後	4.97 (s)	3.54 (s)	2.60 (s)
(速度比)	(1.06)	(1.05)	(1.04)

3.2 プリミティブなオーバーヘッド解析

プログラムを Fleng で記述した場合には, 逐次の C プログラムには不要な処理が埋め込まれることになり, オーバーヘッドを生じる.

そこで, 前述のハンド・コンパイルを行なったプログラムと同等の処理を行なう逐次 C プログラムとを比較し, Fleng の言語特性に起因するオーバーヘッドについて検証を行なった.

それぞれの実行速度を表 3に示す. 本表より, C で記述した場合に比べ, 5 〜 40 倍程度の速度差がオーバーヘッドにより, 生じていることが判った.

この差異の要因を明らかにするため, 最適化後の C プログラムと逐次 C プログラムとの比較を行なった結果, 以下のような差異が明確となった.

1. 変数決定の同期待ち.
   C プログラムでは, 変数値が決定される順番にプログラムを記述するため, 変数値が未決定である状況は存在しない. 一方, Fleng ではデータフロー同期機構による並列動作を実現するために, 変数値が決定されるまで変数束縛の待ち合わせをする必要がある. 変数が未決定の場合, サスペンドが生じ, ゴールフレームの生成といったサスペンド処理が生じ, 高いオーバーヘッドとなる.
2. 動的なメモリ確保.
   C プログラムでは変数領域を静的に割当てて使用するのが一般的である. 一方, Fleng では演算中に必要となった新たな変数に対して, 実行時に動的にメモリに割り付けて使用している. そのため, Fleng プログラムではそのメモリ確保のための処理が必要となる. また, ゴール実行時に GC のチェックを行なう必要も生じる.
3. 算術演算におけるタグの存在.
   C レベルの段階で削除可能なものは取り除いているが, ゴールをまたぐ場合, 各変数にはタグを付けることが必要となる. そのため, C プログラム中では演算が連続して行なえるような場合であっても, Fleng ではタグの着脱作業が必要となる.

表 3: 逐次 C との実行速度の比較

	逐次 C	Fleng(最適化後)
queen (n=11)	0.27 (s)	4.30 (s)
gauss (n=80)	0.07 (s)	0.29 (s)
primes (n=20000)	0.75 (s)	3.78 (s)

*3: 1 スレッド (no lock) で測定

これらのうち, 前二者はメモリ操作を含む処理であり, これがオーバーヘッドの主要因となっている. それぞれによる定量的影響については現在解析中である.

4 おわりに

本稿では, 並列論理型言語 Fleng における実行速度向上を阻害するオーバーヘッドのうち, プリミティブな部分に対する解析を行なった. その結果, 現時点で未実装な最適化により, 平均 10% のコード量削減と 4% の速度向上が得られることが明らかになった. また, 上述の最適化により削減できなかったオーバーヘッドについて解析し, 逐次 C に比べメモリ操作の回数が多いことが主要因であることを考察した. 現在, 実行時間に対する影響の定量的な解析を行なっている.

参考文献

[1] 馬場恒彦, 荒木拓也, 田中英彦. 共有メモリ型並列計算機上の *Fleng* 処理系におけるオーバーヘッドの定量的解析. 情報処理学会第 56 回全国大会, vol.1, pp. 17-18, March, 1997.

[2] 荒木 拓也, 田中 英彦. *Committed-Choice* 型言語 *Fleng* のインライン展開による粒度制御手法. 情報処理学会第 53 回全国大会, vol.1, pp. 347-348, September, 1996.

[3] 荒木 拓也, 田中英彦. *Committed-Choice* 型言語 *Fleng* における静的粒度最適化. プログラミング研究会 96-PRO-8, pp.109 - 114, August, 1996.