5T−1

プログラム実行トレースの高速採取法とその評価

†渡辺 幸光, †林 久美子, †白戸 幸正, ‡妹尾 義樹

†NEC 技術情報システム開発（株）, ‡NEC （株） C&C システム研究所

1 はじめに

アプリケーションのチューニングや性能予測、ボトルネック発見の為にプログラム実行履歴(以下、トレースデータ)を用いるのが一般的である。トレースデータを基にループ長、IF文の真率、各行の実行回数等を知る事が出来、チューニング時には非常に重要な情報となる。しかし従来は、その採取の為に、計測対象プログラム（以下、ターゲットプログラム）に実行履歴計測文を挿入したプログラム（以下、トレースプログラム）を最低1回は実行しなければならないため、採取時の演算量はターゲットプログラムの演算量＋αとなってしまう。

我々は、採取時の演算量をターゲットプログラムの演算量以下に押さえたトレースプログラムを自動生成することにより、トレースデータを高速に採取する方法を開発した。本論文では、高速採取方法の概要(2章)とアルゴリズム(3章)を紹介し、プロトタイプを用いた評価結果(4章)について報告する。

2 概要

採取するトレースデータは、ターゲットプログラムの各基本ブロック[1]実行回数のみとする。よって、トレースプログラムは各基本ブロックをターゲットプログラムと同一回数実行しなければならない。この条件を満たすには、トレースプログラム内の制御文がターゲットプログラムと同一の動作を行えばよい。これは、制御の流れに直接影響する変数(以下、制御変数[2])の値が常に正しければ実現される。したがって制御変数の値に影響を与える文のみ実行させる事により、制御文はオリジナルと同等の動作をする。このトレースプログラムをコンパイル/実行し、高速にトレースデータ採取を行う(図1)。

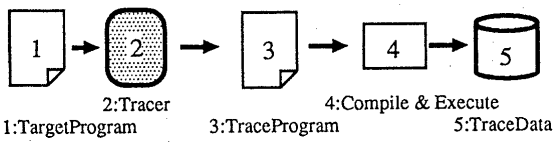

図 1: Tracer システム

3 アルゴリズム

アルゴリズムは、初めにプログラム全体を正しく動作させる為に必要な変数の集合(以下、制御変数集合)を求め(制御変数解析)、ルーチン内に於いて各変数の値を定める可能性のある文集合、いわゆるUSE-DEFチェーン[3]を求める(フロー解析処理)。求められた制御変数集合を出発点とし、USE-DEFチェーンをたどり制御変数集合の値に影響を与える可能性のある文集合を求める(必要文集合解析)。最後に必要文集合の補集合をプログラムから削除し、最適化を行い(最適化)プログラムの各基本ブロックにカウンタインクリメント文、インクリメント変数宣言文、プログラム終了時のカウンタ変数値出力文を挿入する(ソース出力)。

全ての処理はルーチンを単位として行われるが、制御変数解析はプログラム全体として行うため、内部的にサブルーチンコールグラフのボトムアップ、トップダウンの順序で1つのルーチンに対して2度、異なる処理が行われる。図2のコールグラフを用いた場合、初めにSUB3 SUB2 SUB1 MAINの順、そしてMAIN SUB1 SUB2 SUB3の順に処理される。

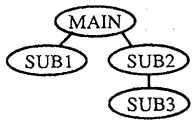

図 2: コールグラフ例

3.1 制御変数解析

あるルーチンで定義された共通変数[4]の値を、他ルーチン内の制御文で引用する場合等は、この引き数を定義している行、およびそれに関連する行を削除する事はできない。このような変数の利用状況を調査する為、初めに各ルーチンの制御変数集合、定義変数集合をもとめる。また、2つの集合の各要素の値に影響を与える変数の集合(以下、依存変数集合)を求め、要素に関連づける。次に各ルーチンが呼び出しているサブルーチンの制御変数集合及び、定義変数集合の共通変数要素に限り、それぞれの集合に足し合わせる。そして、各要素の依存変数集合の共通変数が関連づけられる(以下、STEP1)。集合の足し合わせは以下の規則を用いる。

X: Subroutine$Y$ を呼び出す Routine$X$ の全ての変数集合
Y: Subroutine$Y$ の全ての変数集合
Xa, Ya: X 内,Y 内の制御変数集合
$X'a, Y'a$: X 内,Y 内のルーチンに閉じた制御変数集合
Xb, Yb: X 内,Y 内の定義変数集合
$X'b, Y'b$: X 内,Y 内のルーチンに閉じた定義変数集合
Xc, Yc: X 内,Y 内の制御変数集合
D_α: 変数 α の依存変数集合

$$(Ya \cap X) \cup X'a \rightarrow Xa \quad (1)$$
$$Yb \cap \overline{(X'a \cup X'b)} \rightarrow Xb \quad (2)$$
$$Yb \cap X'a \rightarrow Xa \cap Xb \quad (3)$$
$$Yb \cap X'b \rightarrow Xb \quad (4)$$
$$D_{Ya} \cap X \rightarrow D_{Xa} \quad (5)$$
$$D_{Yb} \cap X \rightarrow D_{Xb} \quad (6)$$

上記処理をコールグラフのボトムアップ順に処理する事により、プログラム入り口ルーチン(例えば図2のMAIN)ではプログラム全体の情報が採取されている。次に、ルーチンコールグラフをトップダウンに処理し、各ルーチンの制御変数集合を求める。ただし、プログラム入り口ルーチンは呼び出しルーチンが無い為、以下の規則に従って入り口ルーチンでの制御変数集合を求め(以下、STEP2)、後述の処理を行う。

$$Xa \cup (Xa \cap Xb) \cup D_{(Xa \cap Xb)} \rightarrow Xc \quad (7)$$

トップダウン処理は呼び出しているサブルーチンの制御変数集合を呼び出し側で求める処理を行う(以下、STEP3)。制御変数集合とサブルーチンの定義変数集合の共通集合及び、サブルーチンの制御変数集合、それぞれの集合要素の依存変数集合をサブルーチンの制御変数集合とする。

$$(Xc \cap Yb) \cup D_{(Xc \cap Yb)} \cup Ya \cup D_{Ya} \cup Yc \rightarrow Yc \quad (8)$$

An Effective Algorithm for Generating Program Execution Traces and Its Evaluation

[1]必ず同一回数実行される、連続した文の集まり

[2]IF文、DO文等の制御文に表れる変数と、その変数を定義する為に引用されている変数

[3]変数の引用、定義の順序、位置を示すチェーン

[4]引き数、CommonItem

例)

```
      PROGRAM EX1
      REAL R2
      INTEGER IA , IB , ICNT
      READ(5,*) ICNT,IA,IB
   10 IF(ICNT .LE. 0) GOTO 99
      CALL SUB(ICNT,IA,IB)
      GOTO 10
   99 R2 = REAL(IA)/REAL(IB)
      WRITE(6,*)'IA =',IA
      WRITE(6,*)'IB =',IB
      WRITE(6,*) 'R2 =',R2
      STOP
      END
c...
      SUBROUTINE SUB(ICNT,IA,IB)
      INTEGER ICNT,IA,IB
      INTEGER ITMP
      ITMP = IA
      IA   = IA + IB
      IB   = ITMP
      ICNT = ICNT - 1
      RETURN
      END
```

上記例で制御変数解析は以下のような処理を行う。

1. SUB のSTEP1 解析結果

   制御変数集合 {}

   定義変数集合 $\{ITMP, IA, IB, ICNT\}$

   依存変数集合の集合

   $\{D_{ITMP} = \{IA, IB\}, D_{IA} = \{IA, IB\},$
   $D_{IB} = \{ITMP, IA, IB\}, D_{ICNT} = \{ICNT\}\}$

2. EX1 のSTEP1 解析結果

   EX1 はサブルーチンSUB を呼び出している為、SUB の各集合をEX1 の集合に足す。SUB には定義集合がある為 式2を用いてEX1 の定義変数集合に足される。足された要素 $IA, IB, ICNT$ は 式6を用いて依存変数集合と関連づけられる。

   制御変数集合 $\{ICNT\}$

   定義変数集合 $\{ICNT, IA, IB, R2\}$

   依存変数集合の集合

   $\{D_{ICNT} = \{ICNT\}, D_{IA} = \{IA, IB\},$
   $D_{IB} = \{IA, IB\}, D_{R2} = \{IA, IB\}\}$

3. STEP2 解析結果

   式7を用いて EX1 での制御変数集合を求める。

   制御変数集合 $\{ICNT\}$

4. EX1 のSTEP3 解析結果

   式8を用いて SUB での制御変数集合を求める。

   SUB の制御変数集合 $\{ICNT\}$

5. SUB のSTEP3 解析結果

   呼び出しサブルーチンなし

3.2 フロー解析

フロー解析により各変数の値を定める可能性のある文集合を求める。

3.3 必要文集合解析

STEP1 でもとめた各ルーチンの制御変数集合を出発点とし、フロー解析で求めたUSE-DEF チェーンをたどり、プログラム全体に影響を与える可能性のある文集合をもとめる。

3.4 最適化

前述処理で求めた文集合の補集合をプログラムより削除し、プログラムの実行における最適化を行う。ただし削除出来る文は代入文のみとする。DO ループが1つの基本Block から構成されている場合で、その内部の演算が全て削除された場合、カウンタインクリメントの為のループになりループ自身に意味がなくなる。この場合DO ループの繰り替えされる回数(ループ長)はDO のパラメータにより計算できる為、DO ループそのものを削除し、ループ長計算式のみ挿入する。また、制御変数集合の補集合の宣言は、それを削除し実行時のメモリ使用量を減らす。制御に影響のない出力文も削除する。

以上の3.1 から3.4 の処理を行った結果得られるプログラムを、前述の例を用いて示す。

```
      PROGRAM EX1
      INTEGER IA , IB , ICNT
      READ(5,*) ICNT,IA,IB
   10 IF(ICNT .LE. 0) GOTO 99
      CALL SUB(ICNT,IA,IB)
      GOTO 10
   99 CONTINUE
      STOP
      END
c...
      SUBROUTINE SUB(ICNT,IA,IB)
      INTEGER ICNT,IA,IB
      ICNT = ICNT - 1
      RETURN
      END
```

3.5 ソース出力

基本ブロック毎にカウンタインクリメント文を挿入した、プログラムの入り口、出口にそれぞれカウンタインクリメント変数宣言文、カウンタ値出力文を挿入し、トレースプログラムを出力する。

4 評価結果

今回プロトタイプ版を作成し、いくつかのプログラムで実験を行った。その結果を以下に示す。
使用マシン：EWS4800/230

Application Name	Target program+α	Trace program
SPIN	5.1(sec)	0.24(sec)
FFT	1.64(sec)	0.06(sec)
LU	301.86(sec)	32.81(sec)

α は実行履歴計測文のオーバーヘッド

以上の結果より、約10倍から20倍の速度向上を達成していることが判る。プロトタイプにおいて本方式の有効性を確認する事ができた。

5 おわりに

本報告では、高速にトレースデータを採取する方法について紹介し、プロトタイプを用いての評価結果を述べた。その結果本方式の有効性を確認する事ができた。今後は多くのアプリケーションに対しての評価を行い、より実用的なシステムに向け改造行う。

参考文献

[1] 佐々政孝: プログラミング言語処理系
岩波講座ソフトウェア科学5、ISBN4-00-010345-8