5Q-02 タイムスタンプを用いたプログラム実行点のポジショニング

丸山 一貴　　寺田 実
東京大学大学院 工学系研究科 情報工学専攻

1 はじめに

プログラムをデバッグするときなど，実行中の場所（実行点）を示すためにソースコード中の位置であるファイル名と行番号が使われる．これは静的な情報であり，実行点の特定には不十分である．

本稿では，タイムスタンプを用いて効率良くプログラム実行点のポジショニングをする方法を述べ，GCC に実装するための修正法を示す．

2 ポジショニングとその応用

図 1に，再帰的に定義された関数 f を持つプログラムの実行過程を示す．図中の (1) で f が呼び出され，その f は再帰によって (2) で自分自身を呼び出す．図中の○印を「関数 f の 7 行目」が実行される場所であるとする．現状では，○印の行が (1) の呼び出しで実行された場合でも (2) の場合でも，共に「7 行目」として示す他なく，繰り返し訪れるそれぞれの場合を識別することが出来ない．

これらの識別，すなわち図 1の○印のどちらかを識別することを実行点のポジショニングと呼ぶこととする．ポジショニングが可能になると，任意の実行点を識別して制御を移動出来るようになり，様々な応用が考えられる．

- デバッグ： 通常のブレークポイントは静的な「ソースコードの行」に対して設定するので，図 1の○印は識別できない．ポジショニングによって，実行点に対してブレークポイントを設定できるようになるので，「図 1の 2 個目の○印でだけ停止する」ことが可能な，動的なブレークポイントが実現できる．
- プログラム理解： ソースコード上では同じ場所でも，特定の実行パスに対してだけコメントを書き込めるようになる．例えば，引数によって異なる処理をする関数であれば，それぞれの処理で呼び出されたときに対応するコメントを書き込み，表示するといった応用が可能である．

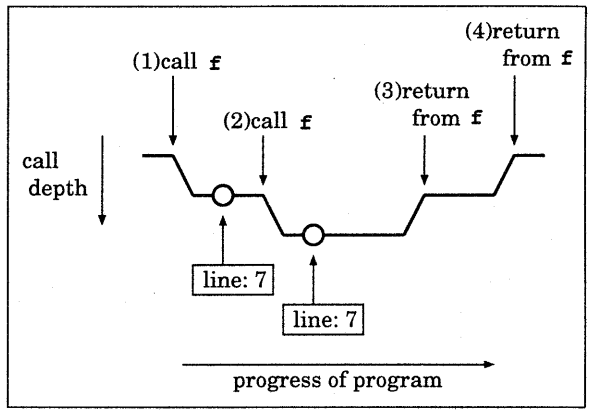

図 1: 実行点の識別

3 タイムスタンプの導入

実行点のポジショニングを実現するには，同じ場所を何度も通過する場合を区別すれば良い．そこでタイムスタンプを変数 timestamp として対象プログラムに導入し，以下の場合にインクリメントする：

- 関数呼び出しの入口・出口
- 関数からの return

Position specification in program traces using timestamps
Kazutaka Maruyama and Minoru Terada
Dept. of Information Engineering, Graduate School of Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo, Tokyo 113-8656, Japan

```
static int timestamp = -1;
static int ref = -1;

void inc_ts_stop(void){}

void inc_ts(void){
  if(++timestamp == ref) inc_ts_stop();
}
```

図 2: タイムスタンプ更新関数

- 上向きの goto
- ループ本体の入口

これにより，ソースコード中のある行を通過するたびに異なるタイムスタンプ値が対応するので，実行点は「タイムスタンプ値」と「行番号」の組で一意に表せる．timestamp のインクリメントは，図 2の関数 inc_ts の呼び出しを対象プログラム中の上記の場所に挿入すれば良い．

4 GCC-2.95 への実装

以上の仕組みを GCC [1] のバージョン 2.95.2 に実装している．GCC は入力を一度，その中間言語である RTL に翻訳してからアセンブリコードに翻訳する．前述の inc_ts の呼び出しは，対象プログラムの RTL を生成する段階で，関数呼び出しを表す RTL として以下の各場所に挿入する：

- 関数呼び出しのプロローグを処理する，関数 start_function の最後
- 関数呼び出しのエピローグを処理する finish_function の先頭
- return を処理する c_expand_return の先頭
- goto を処理する expand_goto 中の，expand_goto_internal を呼び出す直前[1]
- expand_start_loop の，ループ本体の入口にラベルを設定した直後

[1] GCC が独自に拡張している goto のうち，nested function の nonlocal goto には未対応である．computed goto は関数 expand_computed_goto の emit_indirect_jump 呼び出しの直前に挿入すれば良い．

5 関連研究

[2] では，本稿で紹介したタイムスタンプ機構を関数呼び出しの入口と出口だけに挿入することで，大きなオーバーヘッドなく関数単位の逆実行が利用できることを確認している．実装はアセンブリコードレベルで，Intel Architecture に対して行った．

[3] は，本稿のタイムスタンプと似た機構を用いているものの，デバッガの実行制御とデバッガ・デバッギ間通信の削減のため，もっとも細粒度な，ソースコード 1 行ごとのタイムスタンプ更新をしており，通常実行でも 2 倍近い実行時間を要している．

6 まとめ

プログラム実行点のポジショニングを実現する方法として，タイムスタンプを用いて繰り返し実行されるソースコード行を区別する手法を述べ，GCC への実装法を示した．また，この実行点のポジショニングがデバッグやプログラム理解への応用として有用であることを述べた．

参考文献

[1] Stallman, R. M.: *Using and Porting the GNU Compiler Collection*, Free Software Foundation (1999). (included in GCC source code).

[2] 丸山一貴, 寺田実: 関数単位疑似逆実行の高速化, 情報処理学会論文誌 プログラミング (2000). (掲載予定).

[3] Boothe, B.: Efficient Algorithms for Bidirectional Debugging, *Proceedings of the ACM SIGPLAN '00 conference on Programming language design and implementation*, pp. 299–310 (2000).