6L-4

Valgrindを用いたパスプロファイリングツールの開発

大島 一輝 † 大津 金光 † 大川 猛 † 横田 隆史 † 馬場 敬信 †
† 宇都宮大学工学部情報工学科

1 はじめに

プログラムは実行時に挙動が変化する場合があることから, プログラムの実行前に行う静的な最適化には限界がある. よって更なる高速化のためには, 動的にプログラムの最適化を行う必要がある. 動的最適化を行う際には, 実行中のプログラムの挙動を観測するためのプロファイリングが重要となる. 動的最適化を行う上で最も重要な情報の一つは, 複数の基本ブロック間における制御フロー情報である. 本研究では, プログラム全体の制御フロー情報を正確に得るため, 制御フローの各経路を計測するパスプロファイリングに着目した. そこで Valgrind[1] を利用して, Valgrind の計測用コードを挿入するプラグインツールという形でパスプロファイリングの機能を持つツールを開発する. 得られた制御フロー情報はプログラムの動的最適化に活用する. 本稿では Valgrind でパスプロファイリングを実現する方法について述べる.

2 Valgrind

本研究には, 動的計測ツールを構築するためのフレームワークである Valgrind を用いる. Valgrind はオープンソースであり, また x86, AMD64, ARM などの複数の命令セットと, Linux, Darwin (Mac OS X), Android OS に対応している. Valgrind は図 1 で示すように, core とプラグインツールで構成される. core は JIT コンパイラ, 変換コードキャッシュなどで構成され, プログラムのバイナリ変換を行う. プラグインツールはバイナリコードから変換された中間表現コードに計測用コードを挿入する. プラグインツールにより, ユーザーは独自のプロファイリングツールを作成することができる. プロファイリングの対象となるプログラムは以下のような手順で基本ブロックごとに実行される.

(1) Valgrind の core が, プログラムの基本ブロックをマシンコードから中間表現コードに変換する.

(2) 中間表現コードに変換された基本ブロックに, プラグインツールにより計測用コードを挿入する.

(3) 計測用コードが挿入された中間表現コードをマシンコードに変換して実行する.

これらの動作は基本ブロックを単位として繰り返される. 一度変換された基本ブロックは変換コードキャッシュに格納され, 次に同じ基本ブロックを実行するときは変換コードキャッシュ内のバイナリ変換後のコードが実行されるため, 同じ基本ブロックが何度も実行されることで, バイナリ変換処理時に生じるオーバーヘッドの影響は小さくなる. プラグインツールが計測用コードを挿入する対象は中間表現コードであり, 中間表現コードはプロセッサの命令セットに依存しないので, 機械独立な形でパスプロファイリングを行うことができる.

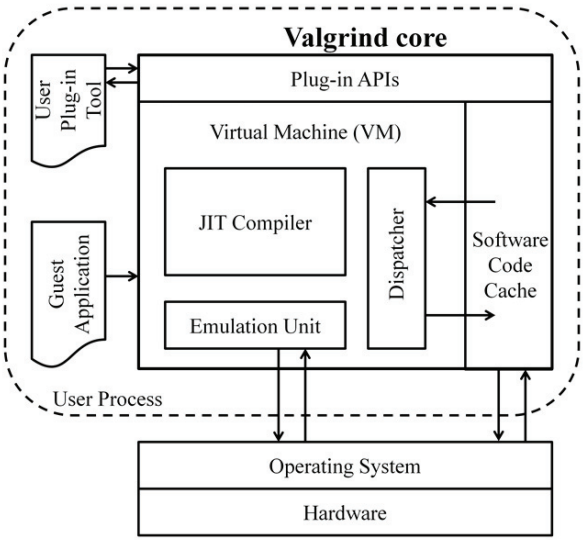

図 1: Valgrind の全体構成

3 パスプロファイリングの実現

3.1 開発方針

本研究で開発するパスプロファイリングツールは, Valgrind で標準的に提供されている, Lackey と呼ばれるプラグインツールを元にする. Lackey は基本的なプログラム計測を行うツールで, プラグインのサンプルとしても使用される. Lackey に備わった機能の中に, 実行された全ての基本ブロックの先頭アドレスを出力するという機能がある. この出力される基本ブロックの先頭アドレスの情報を利用してパスプロファイリングを行う.

3.2 要件

パスプロファイリングツールによって取得する情報として, まずどの基本ブロックがどのような順番で実行されたかを示すパス情報と, それぞれのパスが何回実行されたかという実行頻度がある. 実行頻度の高いパスに最適化を適用することで他の部分より高い効果

Development of Path Profiling Tool using Valgrind
†Kazuki Ooshima, Kanemitsu Ootsu, Takeshi Ohkawa, Takashi Yokota and Takanobu Baba
Department of Information Science, Faculty of Engineering, Utsunomiya University (†)

が得られる．またパスの途中にサブルーチンコールがあった場合, サブルーチンが呼ばれるとその地点でパスを区切り, サブルーチン内のパス情報を, そのサブルーチン内の基本ブロックのみを含むパス情報として管理する方法と, サブルーチンコールでパスが区切られずに, サブルーチン内にも続いたパス情報を管理する方法がある．この 2 つの管理方法を選択可能にする．

3.3 パスの定義

以下の図 2 に制御フローの例を示す．A, B, C, D, E は基本ブロックとする．本研究において, パスは実行した順番に辿る基本ブロックの列とする．バックエッジは分岐先が手前に戻る分岐命令とし, パスはバックエッジで区切る．例の場合, 基本ブロック A は基本ブロック B または基本ブロック C に分岐する．基本ブロック D は基本ブロック A または基本ブロック E に分岐する．基本ブロック D から基本ブロック A に分岐したとき, そのときの分岐命令はバックエッジであるのでパスを区切る．よって図 2 の制御フローには, ABDE, ACDE, ABD, ACD の 4 通りのパスが存在する．

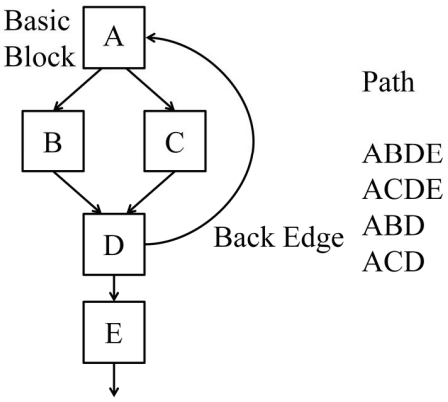

図 2: パスの定義

3.4 データ構造

パスの管理には 2 種類の構造体を用いる．一方の構造体はパスに含まれる基本ブロックの先頭アドレスの列を管理する．各基本ブロックの先頭アドレスはリスト構造により, 実行された順番に連結して管理する．もう一方の構造体は上記のパス情報に対応して実行回数を管理する．各パスの情報はハッシュテーブルに記録して管理することで, パスを検索する際の高速化を図る．

3.5 動作アルゴリズム

パスプロファイリングツールは, 以下のような手順でパスプロファイリングを行う．

(1) 基本ブロックを実行する毎に計測用コードに制御が移る．その基本ブロックの直前に実行した基本ブロックの最後の命令が分岐命令, サブルーチンのコール, リターンのどれであるか判別する．分岐命令であれば (2), サブルーチンのコール, リターンであれば (3) に進む．

(2) (1) で実行した基本ブロックの先頭アドレスの値が, 直前に実行した基本ブロックの先頭アドレスの値より小さいとき, 直前に実行した基本ブロックの分岐命令がバックエッジであるため (3) に進む．小さくなければ (7) に進む．

(3) これまで繋がっていたパスを区切る．

(4) (3) で区切ったパスと同じパスを, これまでに実行しているか検索する．同じパスが見つかった場合は (5), 見つからなかった場合は (6) に進む．

(5) 実行回数をカウントする．(6) を飛ばして (7) に進む．

(6) (3) で区切ったパスを記録する．

(7) (1) で実行した基本ブロックの先頭アドレスをリストに追加する．

(8) 次の基本ブロックに実行が移る．(1) に戻り同様にして動作を行う．全ての基本ブロックの実行が終わった後, 記録したパスと, それぞれのパスの実行回数をファイル出力する．

以上の手順は, サブルーチン内のパス情報を, そのサブルーチン内の基本ブロックのみを含むパス情報として管理するときの手順を示した．サブルーチンコールでパスを区切らないようにする場合は, (1) で実行する基本ブロックの, 直前に実行した基本ブロックの最後の命令がサブルーチンコールとであった場合, (3) ではなく (7) に進むことでパスを区切らないように変更する．

4 おわりに

本稿では, プログラム全体の動的な制御フロー情報を得ることを目的として, Valgrind のプラグインツール Lackey をベースにして開発したパスプロファイリングツールの実現方法を述べた．

謝辞

本研究は，一部日本学術振興会科学研究費補助金 (基盤研究 (C)24500055，同 (C)24500054) の援助による．

参考文献

[1] Nicholas Nethercote and Julian Seward: “Valgrind: A Framework for Heavyweight Dynamic Binary Instrmentation”, Proc. of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), pp.89-100, 2007.

[2] Thomas Ball and James R. Larus: “Efficient Path Profiling”, Proc. 29th Ann. IEEE/ACM Int’l Symp. Microarchitecture, pp.46-57, 1996.