動的最適化システムにおける効率的なプロファイリング手法
An Efficient Profiling Technique for Dynamic Optimizing System

岩本 智志 † 古関 聰 ‡ 小松 秀昭 ‡ 深澤 良彰 †
Satoshi Iwamoto Akira Koseki Hideaki Komatsu Yoshiaki Fukazawa

1. はじめに

動的最適化システムにおいて、実行時のプロファイリングによって得られた情報をもとに動的に最適化を行うことで大きなパフォーマンスの向上が期待できる。

効果の高い最適化を行うためには詳細なプロファイル情報が必要となるが、プログラム全体に対して継続的に詳細なプロファイリングを行った場合、そのオーバーヘッドが大きくなり、最適化によって高速化しても全体としての速度向上は難しい。

プロファイリングを 2 段階にすることでオーバーヘッドを削減する手法 [1] がある。この手法は、まず、おおまかなプロファイリングによって頻繁に実行される部分（ホットスポット）を検出する。さらに、検出されたホットスポットについて詳細なプロファイリングを行うようにプロファイリングのモードを切り替えることで低コストのプロファイリングを実現する。しかし、ホットスポットを抽出する方法としてサンプリングを利用しているために軽量である反面、ホットスポット抽出の精度が悪くなることがある。また、複数の種類のプロファイリングを行うことが出来ないという問題がある。

また、コードを複製し、オリジナルコードとプロファイルコードを挿入したコードを切り替えることでオーバーヘッドを削減する手法 [2] がある。実行中にコード書き換えを行わないためアーキテクチャに依存せず、両者のコードのスイッチングのオーバーヘッドが小さいという利点があるが、ホットスポットにおけるコード量が増加してしまいパフォーマンスの低下を招くおそれがある。

そこで本手法では、まず、軽量なプロファイルコードの挿入によってホットスポットを検出する。検出されたホットスポットに対して詳細なプロファイリングを行うように実行中にコードを書き換えることで、軽量なプロファイリングと詳細なプロファイリングのモードの切り替えを行う。

本手法には、以下のような特徴がある。

- カウンタを利用した正確なプロファイリング。
- プロファイルコードの挿入によりプロファイリングのきめ細かい制御が可能。
- 動的なコードの書き換えにより実行中に詳細なプロファイリングと軽量なプロファイリングを実行中に切り替えることが可能。多段階に切り替えてもコード量がほとんど増加しない。

†早稲田大学大学院理工学研究科
Graduate School of Science and Engineering, Waseda Univ.
‡日本 IBM (株) 東京基礎研究所
Tokyo Research Laboratory, IBM Japan, Ltd.

2. プロファイリング手法

本手法は、チェックコードによってホットスポットを検出し、検出されたホットスポットに対してプロファイリングモードの切り替えを行うことで実現する。

2.1 チェックコード

ホットスポットを検出しプロファイリングモードを切り替えるコードをチェックコード (図 1) と呼ぶ。チェックコードは挿入した部分に対してプロファイルを取るかどうかを判断し、その切り替えを行う。チェックコードは全てのバックエッジごとに挿入される。

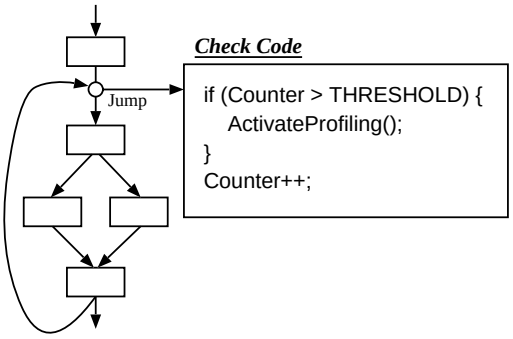

図 1: チェックコード

詳細なプロファイリングは最適化の効果が高いホットスポットに対して行うことが望ましい。ホットスポット検出の手順を以下に示す。チェックコードはこのコードを実行した回数を計測し、その回数が閾値を超えると詳細なプロファイリングを行うようにモードを切り替える (ActivateProfiling)。閾値未満ならば詳細なプロファイリングは行わない。

また、必要なプロファイリングを行ったらパフォーマンスの低下を防ぐためにプロファイリングを止める必要がある。よって、閾値が一定回数を超えたらプロファイリングを止めるようにする。

プロファイリングモードを切り替える条件はこれに限らず、多段階の閾値を設定して徐々に詳細なプロファイリングを行うようにするなど、 自由に条件を設定することが可能である。

2.2 プロファイリングモードの切り替え

プロファイリングモードを切り替える方法を図 2 に示す。あらかじめ、詳細なプロファイリングを行う場所に NOP(No Operation) 命令を挿入しておく。NOP 命令は実際には何も行わない冗長な命令である。この NOP 命令を詳細なプロファイルコードへの分岐命令に置換することで詳細なプロファイリングを行うモードに切り替え

ることができる。詳細なプロファイリングを止めるには分岐命令を元の NOP 命令に戻せばよい。

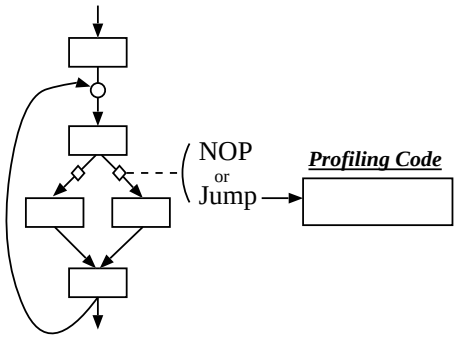

図 2: プロファイリングモードの切り替え

置換するためには、チェックコードに対応するループ内の NOP 命令を参照できるようにする必要があるので、コンパイルの際にチェックコードに対応する NOP 命令のアドレスを関連付けて保存しておく。モードの切り替えを行うには、関連付けの情報を参照して順に NOP 命令をプロファイルコードへの分岐命令に置換すればよい。

3. 実験

本手法を実装と評価を行い、プロファイリングのオーバーヘッドを削減できることを示す。

3.1 方法

評価には Daisy[3] 上にプロファイルを取る機能を追加したものを用いた。Daisy は PowerPC マシンコードを入力とした仮想マシンとして動作するものであり、動的最適化システムのシミュレーション環境として利用する。

評価用アプリケーションには SPECint2000 の 164.gzip, 300.twolf を用い、gcc 2.95.3 -O3 でコンパイルした。入力データセットは test とした。評価用ハードウェアは IBM RS/6000, CPU: powerPC 604e 250MHz, メモリ: 128MB, OS: AIX 4.3 を用いた。

3.2 プロファイリングのオーバーヘッドの比較

評価はエッジプロファイリングを行い、そのオーバーヘッドを計測した。全ループに対してエッジプロファイリングを行ったものと、本手法を利用してホットスポットのみに対してエッジプロファイリングを行ったものとのオーバーヘッドを比較した。また、閾値を変化させた場合のオーバーヘッドを比較した。

表 1 に本手法を適用したプロファイリングのオーバーヘッドを示す。数値はプロファイルを取らないコードの実行速度を 1 としたときの比率である。モード切り替えの閾値は 30000 とした。

表 2 は閾値とオーバーヘッドの比較である。表の「常にプロファイリング」は全てのループに対して常に詳細なプロファイリングを取った場合のオーバーヘッドである。

3.3 考察

表 1 から、本手法を適用して閾値を適切に設定することで軽量なプロファイリングを実現できることが分かる。

また、表 2 から、閾値を大きくしていくことで詳細なプロファイリングをホットスポットに限定し、オーバーヘッドを削減できることが分かる。ただし、閾値を上げすぎるとホットスポット抽出の遅延が大きくなってしまうため適切な閾値を設定する必要がある。

閾値を 100 とした場合のオーバヘッドが常にプロファイリングを行った場合より 10%程度増えている。これはプロファイルモードの切り替えのオーバーヘッドであるが、大部分のループに対して切り替えを行った場合のオーバーヘッドであるので、十分軽量であると考えられる。

全体的にオーバーヘッドが高めとなっているが、これは本手法によるオーバーヘッドの削減の評価を行うために重めのプロファイリングの実装としたためである。

表 1: プロファイリングのオーバーヘッドの比較

ベンチマーク	本手法	常にプロファイリング	比率
164.gzip	11.88	15.43	0.770
300.twolf	9.45	11.59	0.815

表 2: 閾値とプロファイリングのオーバーヘッド

閾値	オーバーヘッド
(常にプロファイリング)	15.51
100	16.01
1,000	15.65
10,000	13.79
100,000	9.24

4. おわりに

本稿ではコードの挿入によるプロファイリングを低コストで行う手法を提案した。本手法はプロファイルコードの挿入による 2 段階のプロファイリングを行い、動的にコードを書き換えることでプロファイリングモードを切り替える。本手法を適用することで、負荷の高いプロファイリングを必要最小限のオーバーヘッドで行うことが可能となる。

参考文献

[1] M. Burrows, P. Erlingson, S.-T. S. Leung, M. T. Vandevoorde, C. A. Waldspurger, K. Walker, and W. E. Weihl. Efficient and Flexible Value Sampling. ASPLOS 2000.

[2] Matthew Arnold, and Barbara G. Ryder. A Framework for Reducing the Cost of Instrumented Code. PLDI 2001.

[3] Erik R. Altman, and Kemal Ebicioglu. “DAISY Dynamic Binary Translation Software”. IBM T.J. Watson Research Center, 2000.