2A-2

特定のメモリ領域のアドレストレースを採取するソフトウェアライブラリの実装*

城勘 友秀† 白木原 敏雄†
株式会社 東芝 研究開発センター‡

1. はじめに

アドレストレースとは、プロセスが実行中にメモリ領域のどのアドレスにアクセスしたかの履歴である。コンパイラの分岐予測の解析、キャッシュ構成の探索など、ソフトウェアやハードウェアの最適化のためのプロファイルデータとして頻繁に利用される。

プロセスがアクセスする全てのメモリ領域を対象とするアドレストレースは、従来から存在する方法により採取可能である。しかしマルチメディア処理など、大量のデータ処理を行うプロセスのアドレストレースは膨大な量であり、評価に必要な部分を抽出するのに時間がかかる。そこで我々は、特定のデータ領域のみのアドレストレースを採取する方法を提案し、ライブラリとして実装した。本論文は、この方法と、実装したライブラリの性能評価について述べる。

2. 従来方法

従来から、いくつかのアドレストレース採取方法が提案されている [1]。ここでは我々の提案方法と最も類似する方法を挙げる。それは、UNIX システムで一般的に使用される、GNU GDB のようなデバッガの手法を応用する方法である [2]。この方法は、まずデバッガプロセスがターゲットプロセスをアタッチする。そして、デバッガプロセスが、アドレス空間操作、シグナル横取り、実行操作をターゲットプロセスに対して行い、アドレストレースを採取する。

この方法による、具体的なアドレストレース採取の流れを図 1 に示す。インストラクション (図 1 内の A) のメモリ領域へのアクセスを検出するには、まず `mprotect` システムコールによりそのメモリ領域をプロテクトする。そして、発生したセグメンテーションフォールト (`SIGSEGV`) を横取りする。`SIGSEGV` の例外処理では、例外からの復帰後に A を実行可能にするためにプロテクトを解除する。しかしそのまま実行を続けると、A がアクセスしたメモリ領域へのアクセスは、再度検出できなくなる。そこで、シングルステップ実行に切替えてから A の実行に復帰し、A の実行完了を `SIGTRAP` により把握する。このシグナルも横取りし、例外処理内で再びメモリ領域をプロテクトする。

この方法は、オーバヘッドとして例外処理への遷移が最低 2 回発生し、さらに、例外処理はデバッガプロセスで行うために、プロセス切替えが最低 4 回発生する。

3. 提案方法

提案方法は従来方法と類似しているが、従来方法との違いは次の 2 つである。第 1 に、ライブラリとして実装し、例外処理を 1 つのプロセス内で行うことで、プロセス切替えを不要にする。第 2 に、例外処理内でセグメンテーションフォールトを起こしたインストラクションを模擬実行し、次のインストラクションに復帰する。これにより、例外処理への遷移を 1 回に抑えられる (インストラクション模擬実行については、3.2. 節で詳しく述べる)。

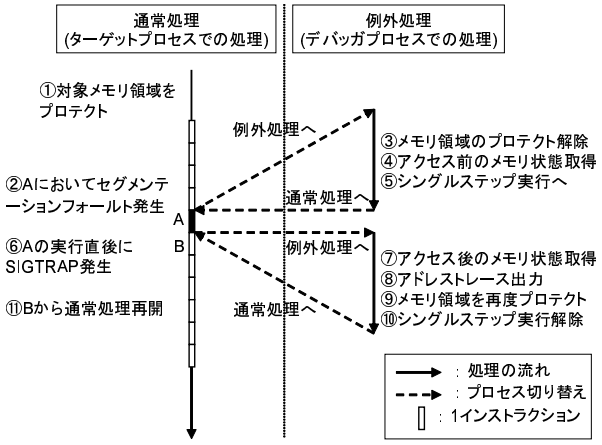

図 1: 従来方法によるアドレストレース採取の流れ

3.1. 処理の流れ

提案方法によるアドレストレース採取の流れを図 2 に示す。また、具体的な処理の手順は、以下の通りである。

1. ユーザがアドレストレース採取を開始する際に、アドレストレースを採取するメモリ領域をプロテクト (`mprotect` システムコール)。
2. プロセスが、プロテクトされているメモリ領域へアクセスするインストラクションを実行しようとすると、セグメンテーションフォールト (`SIGSEGV`) が発生。ハンドラがこれを捕捉し、例外処理へ遷移。
3. セグメンテーションフォールトが発生したメモリ領域のプロテクトを解除 (`mprotect` システムコール)。
4. セグメンテーションフォールトを発生させたインストラクションを、模擬実行。
5. そのインストラクションがアクセスしたメモリ領域のアドレスを把握し、アドレストレースを出力。
6. 3 でプロテクト解除したメモリ領域を再度、プロテクト。
7. 通常処理へ復帰し、セグメンテーションフォールトを発生させたインストラクションの次のインストラクションから実行を再開。

*Implementation of a Software Library for Capturing Address traces on the Specific Memory Area.
†Tomohide JOKAN and Toshio SHIRAKIHARA
‡Corporate Research and Development Center, TOSHIBA Corporation

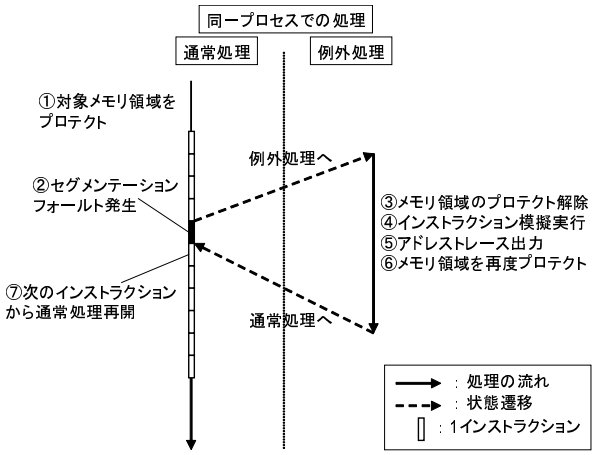

図 2: 提案方法によるアドレストレース採取の流れ

3.2. インストラクション模擬実行

インストラクション模擬実行とは、セグメンテーションフォールトを発生させたインストラクションのみを、例外処理の中で実行することである。メモリ領域や通常処理のコンテクスト内のレジスタ値を更新することで、通常処理への復帰後、そのインストラクションが実行された直後の状態になる。処理の手順を以下に示す。

1. インストラクションをデコードし、どのような処理をすべきか把握。
2. 通常処理のコンテクストから、レジスタ値を取得。
3. メモリ値を取得。
4. 2 と 3 の値を使い演算を実行。
5. 演算に伴うフラグ処理を行い、通常処理のコンテクストのフラグレジスタ値を更新。
6. メモリ値を更新。
7. 通常処理のコンテクストのレジスタ値を更新。
8. 通常処理のコンテクストのプログラムカウンタを、次に実行すべきインストラクションへ進める。

4. ライブラリの実装

提案方法を、IA32 Linux 上で動作するライブラリとして実装した。従来方法と提案方法のプロセス構成を図 3 に示す。本ライブラリ内には、`SIGSEGV` ハンドラが含まれており、ターゲットプログラムの例外処理をライブラリ内で行う。例外処理を同一プロセス上で行うため、プロセス切替えが不要となる。

アドレストレースを採取するには、まず本ライブラリをターゲットプログラムにリンクする。そして採取ターゲットのプログラムから各種ライブラリ関数を呼び出す。本ライブラリに用意した主な関数とその機能は、以下の通りである。

- `memtr_init(void)`：ライブラリの初期化
- `memtr_regist(void *addr, size_t len)`：アドレストレース採取対象とするメモリ領域の指定
- `memtr_remove(void *addr)`：指定されているアドレストレース採取対象のメモリ領域を、指定解除
- `memtr_end(void)`：プログラム終了前の処理

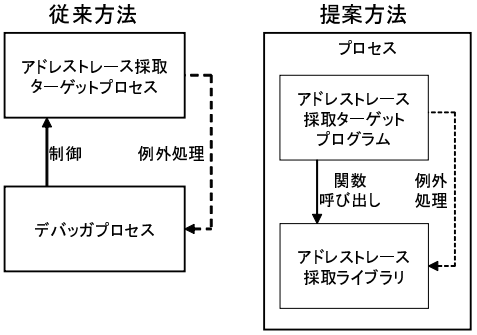

図 3: 従来方法と提案方法のプロセス構成

表 1: ライブラリの処理時間

処理内容	処理時間 (秒)	処理回数 (回)
コンテクスト スイッチング	146.0	33,650,561
`mprotect` システムコール	133.9	33,650,561
その他	35.2	–
全体	315.1	–

5. 性能評価

実装したライブラリを、フリーの H.264 エンコーダである x264[3] へ適用した。そして、x264 で VGA サイズの動画 1 フレーム分のエンコードを実行させてアドレストレースを採取し、その時のライブラリの処理時間を、`getrusage` 関数を使って計測した。結果を表 1 に示す。

最も時間を消費している処理は、通常処理から例外処理への遷移の際に行われる、コンテクストスイッチングである。次いで、メモリ領域のプロテクト及びプロテクト解除で呼び出される `mprotect` システムコールである。

6. まとめと今後の課題

本論文では、特定のメモリ領域のみのアドレストレースを採取する方法を提案し、実装方法を示した。性能評価の結果、例外処理やメモリプロテクションなどのシステム時間が大きいことが分かった。今後は他の方法との性能比較を行い、有用性を確認していく。

参考文献

[1] A. Agarwal, R. L. Sites, and M. Horowitz, “ATUM: A New Technique for Capturing Address Traces Using Microcode,” Proc. 13th International Symposium on Computer Architecture, pp.119–127, June 1986.

[2] 中村明, 相田仁, 計宇生, 小池汎平 共訳, “UNIX 4.3BSD の設計と実装,” 丸善株式会社, 1991.

[3] “VideoLAN - x264,” http://developers.videolan.org/x264.html.