メモリバグ検出ツールMEMLIGHTのバグ検出高速化技術

5D－10

杉森 英夫 †

†住友金属工業(株)

1 はじめに

我々はMMU(Memory Management Unit)機能を利用したメモリバグ検出ツールMEMLIGHTを開発した。一般的にメモリバグ検出機構が組み込まれたプログラムは、実行時に生じる動的メモリに対するメモリアクセスをチェックすることによってバグ検出を行なっている。実行時のメモリアクセスチェックのオーバーヘッドはプログラムの実行性能に大きく影響する。そこで、メモリアクセスのチェック手法の最適化を行なうことにより、メモリバグ検出のオーバーヘッドを小さくできる。

本稿では、メモリバグ検出精度を低下させず、かつメモリバグ検出のオーバーヘッドを小さくするための最適な検出方式について述べる

2 MEMLIGHTのメモリバグ検出

MEMLIGHTのメモリバグ検出の基本原理ではMMUの機能を利用している。プログラムは、MEMLIGHTの動的メモリ割当機能を用いて割当てられているアクセス不可領域を動的メモリと同様にアクセスを行なう。アクセス不可領域のメモリにアクセスが行なわれると、そのメモリにはアクセス権が無いためにMMUがハードウェア割り込みを発生させ、事前に登録されている割り込みハンドラが呼び出される。割り込みハンドラでは、動的メモリにアクセスを行なった命令の解析を行ない、アクセスの対象となったアドレスを抽出して、そのアドレスがアクセス有効範囲であればアクセスは正当であるとみなし実メモリアドレスに対するメモリアクセス命令の代行を行なう。不当であるなら警告を出す。

アドレスがアクセス有効範囲であるかどうかをチェックするためには大きなアドレス変換テーブルを検索するため、チェックのオーバーヘッドは大きくなってしまう。また、MMUからの割り込みを受け取るハンドラでメモリアクセスのチェックを行なうため割り込み発生時のコンテキストの切替えなどにより、メモリバグ検出のオーバーヘッドは大きくなる。そこで、メモリバグ検出のためのコストを小さくする方法として、次のような高速化手法を検討した。

- アドレスキャッシュを用いた最適化
- 動的テキスト書き換えによる最適化

2.1 アドレスキャッシュを用いた最適化

メモリバグ検出は、割り込みハンドラで、メモリアクセスを行なった命令の解析を行ない、アクセスの対象となったアドレスを抽出し、アドレス変換テーブルを検索して、そのアドレスが、そのアクセスが正当なアクセスがどうかを判断している。この方法では、単一の動的メモリブロックに対して正当なメモリアクセスが何度も行なわれる時、命令が実行される度にハンドラ内で大きなアドレス変換テーブルを検索するためオーバーヘッドがかかってしまう。

そこで、図1の様に以前一度検索されたメモリの有効範囲、実メモリとのアドレスオフセット値をキャッシュバッファに入れておく。実メモリのアドレスは、アクセスされたアドレスにこのオフセット値を加えて計算される。アドレス変換テーブルを検索する前にキャッシュバッファを検索してヒットした時は、キャッシュのオフセット値より実メモリのアドレスを算出する。ヒットしなければ従来の方式どおりアドレス変換テーブルを検索する。

	アクセス不可領域先頭アドレス	アクセス不可領域終了アドレス	実メモリアドレスオフセット
cache[0]			
cache[1]	⋮	⋮	⋮
cache[2]	⋮	⋮	⋮
cache[3]	⋮	⋮	⋮

図1: アドレスキャッシュバッファ

キャッシュバッファを用いることによって、一つの動的メモリブロックに対して連続してメモリアクセスが行なわれる場合など、そのメモリアクセスが正当なものである時、キャッシュバッファにアドレスの有効範囲を入れておくことによりメモリアクセスのチェックを

Hideo SUGIMORI†
†SUMITOMO Metal Industries, LTD.

簡略化できる。これによって、アドレス検索、変換のコストを小さくできる。

2.2 動的テキスト書き換えによる最適化

MMU の機能を利用する場合 MMU からの割り込みを受け取った割り込みハンドラで、メモリアクセスのチェックを行なうことになる。この方法では、動的メモリにアクセスが行なわれる度に MMU からの割り込みが発生してしまうため、その都度発生するコンテキストの切替えによって、メモリバグ検出のオーバーヘッドはかなり大きくなってしまう。

そこで、割り込みハンドラで行なう動的メモリアクセスに対するアドレス変換、キャッシュバッファ検索部分を、最初に MMU から割り込みを受け取った時点でパッチコード (機械語命令列) としてデータエリアに作成する。(図 2) パッチコードには、命令解析を行なった結果を反映させるため、メモリアクセス命令につき一つのパッチコードを持つ。図 2 のようにデータ域に生成するパッチコードは、次の部分からなる。

1. レジスタのセーブ
2. メモリアクセス命令の解析
3. アドレスバッファの検索
4. アドレステーブル検索、アクセスチェック手続き呼び出し
5. メモリアクセス命令代行
6. レジスタリストア
7. メモリアクセス命令の次のアドレスへ分岐

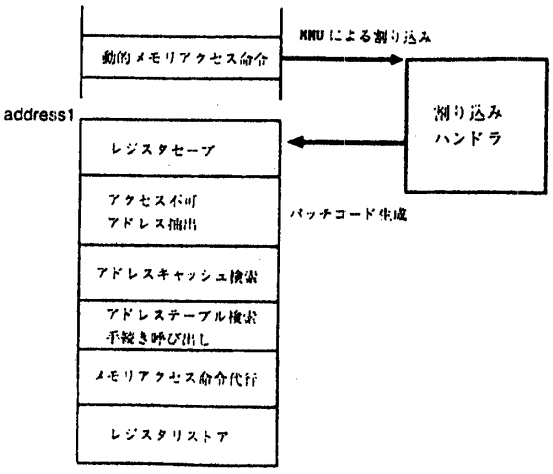

図 2: パッチコードの生成

動的メモリに対するアクセスを行なう命令を、そのデータエリアに作成した命令列への分岐命令に置き換える。パッチコードを生成した後、割り込みハンドラより復帰して生成されたパッチコードを実行する。パッチコードでは、入口ですべてのレジスタをセーブして、出口でリストアしているので、プログラム実行上の問題は出ない。2 度目以降にその動的メモリアクセスを行なう命令を実行する時は、直接パッチコードへ分岐してパッチコードを実行する。(図 3)

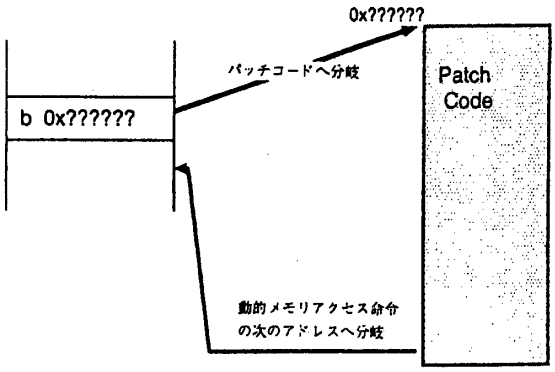

図 3: パッチコード生成後の実行イメージ

実行ループなどによって、何度も同じメモリアクセス命令が実行される場合など、MMU からの割り込みの発生は最初の 1 回だけで済む。2 回目以降のアクセスチェックはパッチコード内部で行なわれるため、ハードウェア割り込みによるオーバーヘッドはほとんど無視できる。これによって MMU からの割り込みを発生させずにメモリアクセスのチェックを行なうことが可能であるため、割り込み発生時のオーバーヘッドを小さくできる。

遅延スロットを持つ RISC プロセッサの場合、メモリアクセスを行なう命令が遅延スロット内にある場合、パッチコードの生成はその遅延スロットを考慮した形にする必要がある。

3 おわりに

本稿では、メモリバグの検出精度を低下させず、メモリバグ検出のオーバーヘッドを小さくするための最適化について述べてきた。アクセスされたアドレスをキャッシングする方式と、動的にテキストを書き換え、データエリアに生成された命令列を実行させることにより、メモリバグ検出のオーバーヘッドをかなり低減できる方式を明らかにした。