2ZB−01

サイレントストアを利用したメモリバイオレーションの削減

峯 博史† 服部 直也‡ 坂井 修一‡ 田中 英彦‡
†東京大学大学院工学系研究科 ‡東京大学大学院情報理工学系研究科

1 はじめに

スレッドレベルの投機実行を行うアーキテクチャ[1]において、投機スレッドによるメモリの投機的アクセスは、パフォーマンスの向上を図るために必要不可欠な技術である。しかし、プログラムの処理の正当性を保証するためには、投機スレッドによるスレッド間のメモリ依存を侵すメモリアクセスをメモリバイオレーションとして検出し、その投機スレッドを破棄する機構[2]が必要である。バイオレーションのために破棄されたスレッドの処理に費やされたサイクル数は、純粋に浪費された計算資源であり、メモリ投機ミスによるメモリバイオレーションの頻発はパフォーマンスに深刻な影響を及ぼす。

本稿では、同じデータを上書きするストアであるサイレントストア[3]をバイオレーションの判定時に考慮することで、メモリバイオレーションを削減できる可能性があることを述べる。

2 投機を支援するキャッシュ機構

投機的メモリアクセスを支援する機構として、スレッドの処理単位に備わるプライベートな一次キャッシュのキャッシュプロトコルを拡張する手法[4]が提案されている。この機構をさらに拡張し、アップデート型のキャッシュに適用することにより、バイオレーションの判定時にサイレントストアを考慮することができる。

2.1 基本機構

各キャッシュラインは、有効(Val)、無効(Inv)に加えて、投機スレッドによる投機的メモリアクセスを記録するために、投機ロード(SpL)、投機ストア(SpS)の状態をとる。ラインサイズがメモリアクセスの最小単位の語長を超える場合はフォールスシェアリングと同様の問題が生じるため、状態遷移には場合分けが必要である。(表1)

バイオレーションの検出は、親スレッドがストアを行った場合にメモリアドレスを取得し、対応するキャッシュラインの状態を調べることによって行われる。状態が、(A)ならSpLの場合に、(B)ならSpLまたはSpSの場合に、それぞれメモリバイオレーションであると判定される。

表1: 基本状態遷移

(A)single-word line

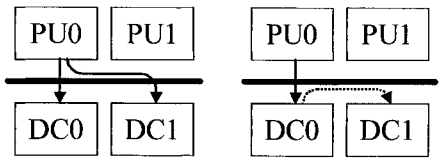
	LD	ST
Inv	SpL	SpS
Val	SpL	SpS
SpS	SpS	SpS
SpL	SpL	SpL

(B)multi-word line

	LD	ST
Inv	SpL	SpS
Val	SpL	SpS
SpS	SpS	SpS
SpL	SpL	SpS

2.2 サイレントストアを考慮する機構

サイレントストアを考慮する場合、(a)全てのストアをバスに流し、各キャッシュで判定を行う、(b)ストアが行われるキャッシュで判定を行い、それがサイレントストアでない場合にバイオレーションの判定のためにバスに流す、の二通りが考えられる。(図1)



(a) ブロードキャスト　(b) フィルタリング

図1: サイレントストアの検出場所

(a)では、ストアを行う度にバスを使う必要があるとともに、各キャッシュにおいて、投機ロード時のデータを保持しておくために投機ロードが行われたキャッシュラインに対するストアはストールさせる必要がある。それに対し、(b)では、各キャッシュで投機ロードを行ったラインに対するストアをストールさせる必要がなく、ストアがサイレントストアであった場合にはバスを占有せずに済むという利点がある。よって、本稿では(b)の場合について評価を行う。

3 評価

サイレントストアをバイオレーションの判定時に除外することによる、メモリバイオレーションの削減について評価するために、SPECint95の8種のベンチマークプログラムを対象とするトレースベースのシミュレーションを行った。

Memory Violation Reduction with Silent Store
Hiroshi MINE†, Naoya HATTORI‡, Shuichi SAKAI‡, Hidehiko TANAKA‡ {mine,hato,sakai,tanaka}@mtl.t.u-tokyo.ac.jp
†Graduate School of Engineering, The University of Tokyo
‡Graduate School of Information Science and Technology, The University of Tokyo

3.1 シミュレーションモデル

スレッドレベルの投機実行を行うアーキテクチャとして、図2のようなチップマルチプロセッサを仮定した。各プロセッシングユニット(PU)には、メモリ投機を支援するプライベートな一次キャッシュとレジスタ通信のための機構が備わっているものとし、スーパースカラコアはAlpha21264相当とした。

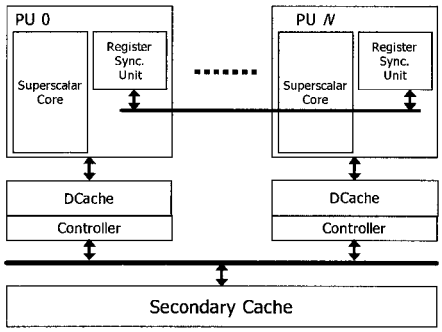


図2: アーキテクチャモデル

プログラムのスレッド分割は、コンパイラによって自動的に行うが、スレッド化されていないライブラリ関数の影響を回避するため、該当部分は実行トレースより除外した。

4PU構成で、一次キャッシュは、データ容量8Kバイト4ウェイLRU、メモリアクセスレイテンシはヒット時1サイクル、ミス時6サイクルとし、ラインサイズが、(A)1語-8バイト、(B)8語-64バイトの二通りについて、単純にメモリ投機を行った場合と、バイオレーションの判定時にサイレントストアを考慮した場合とを比較した。

なお、制御予測は完全とし、共有二次キャッシュは100%ヒットとした。

3.2 結果と考察

(A)、(B)それぞれについて、サイレントストアを考慮することで、バイオレーションの発生を削減できた割合を図3に、その時の速度向上を図4に示す。

サイレントストアをバイオレーションの判定時に考慮することで、合計で(A)約62%、(B)約47%のバイオレーションの発生を削減することができ、調和平均で(A)約7%(B)約6%の速度向上を得ることができた。

(A)のliでバイオレーションの発生回数が増えているが、これは、サイレントストアを考慮することで、本来破棄されるべきスレッドのバイオレーション検出のタイミングが遅れ、それに後続するスレッドのバイオレーションの発生条件が変化したためだと考えられる。

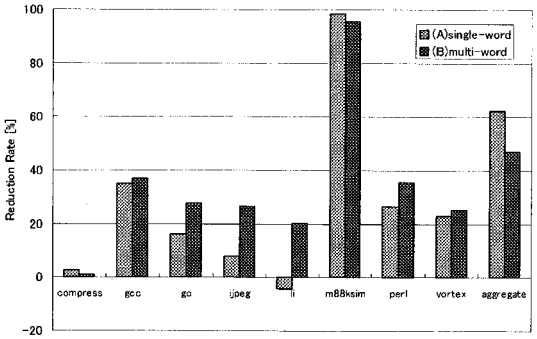


図3: バイオレーション発生回数の削減率

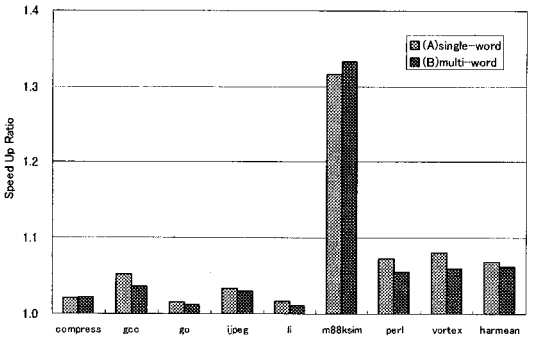


図4: 速度向上

4 おわりに

本稿では、バイオレーションの判定からサイレントストアを除外することでメモリバイオレーションの発生回数を削減し、投機スレッドの破棄に伴うパフォーマンスペナルティを回避できる可能性があることを述べた。

今後は、キャッシュのラインサイズがパフォーマンスに与える影響についての研究を行う。

参考文献

[1] Krishnan, V. and Torrellas, J.: A Chip-Multiprocessor Architecture with Speculative Multithreading, *IEEE Transactions on Computers*, Vol. 48, No. 9, pp. 866 – 880 (1999).

[2] Gopal, S., Vijaykumar, T. N., Smith, J. E. and Sohi, G. S.: Speculative Versioning Cache, *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pp. 195 – 205 (1998).

[3] Lepak, K. M. and Lipasti, M. H.: On the value locality of store instructions, *Proceedings of the 27th International Symposium on Computer Architecture*, pp. 182 – 191 (2000).

[4] Steffan, J. G., Colohan, C. B., Zhai, A. and Mowry, T. C.: A scalable approach to thread-level speculation, *Proceedings of the 27th International Symposium on Computer Architecture*, pp. 1 – 24 (2000).