5K-1

複数トランザクション間の競合を考慮したLogTMにおけるアボート対象選択手法

堀場 匠一朗† 江藤 正通† 浅井 宏樹† 津邑 公暁† 松尾 啓志†
†名古屋工業大学

1 はじめに

マルチコア環境における並列プログラミングでは，共有リソースへのアクセス制御にロックが広く用いられている．しかし，ロックには並列性の低下やデッドロックの発生などの問題がある．そこで，ロックを用いない並行性制御機構として Transactional Memory(TM)[1] が提案されている．TM の一実装である LogTM[2] ではトランザクションが投機的に実行され，possible_cycle フラグと呼ばれるフラグを用いてデッドロックの発生を検出する．しかし，この手法では，実際にはデッドロック状態にない場合でもデッドロックとして検出されアボートが過剰に発生する可能性がある．そこで，3 者以上のトランザクション間の依存関係を考慮し，デッドロックを検出可能にした上で適切なアボート対象を選択する手法を提案する．

2 背景

2.1 トランザクショナル・メモリ

TM は，データベースにおけるトランザクション処理をメモリアクセスに適用したものであり，複数トランザクションによる同一アドレスへのアクセスを競合として検出する．競合を検出した場合，どちらかのトランザクションの実行を中断し，それまでの実行結果を破棄する．これをアボートという．一方競合が検出されない場合，実行結果を元のメモリアドレスへ反映させる．これをコミットという．この TM をハードウェア上に実装したもののひとつに，LogTM がある．

2.2 LogTM

2.2.1 データのバージョン管理

投機的実行では実行結果が破棄される可能性があるため，LogTM はアクセスするデータの古い値を，ログ領域に保持し管理する．投機実行が成功した場合，トランザクションをコミットするが，実行結果は既にメモリに反映されているため，ログの内容を破棄するだけでよい．一方投機実行が失敗した場合，トランザクションをアボートし，ログに保存された値を元のメモリアドレスに書き戻すことで，トランザクション開始時点のメモリ状態を復元する．

A Method for Aborting Transactions by Considering Deadlock Loops
†Shoichiro HORIBA †Masamichi ETO †Hiroki ASAI †Tomoaki TSUMURA †Hiroshi MATSUO

2.2.2 競合検出

LogTM では Illinois プロトコル [3] を拡張したキャッシュ一貫性モデルを採用している．Illinois プロトコルでは，メモリへアクセスするスレッドはキャッシュコヒーレンスリクエストを各スレッドに送信する．LogTM では，このリクエストを受信した時，各スレッドが競合の発生を検査している．これは，トランザクション内で発生した R/W アクセスを記憶するためのビットをキャッシュライン上に追加し，そのビットを参照することで実現される．競合が発生しなかった場合，リクエストを送信したスレッドへ ACK が返信される．一方競合が発生した場合には，NACK が返信される．NACK を受信したスレッドは競合したトランザクションが終了するまで一時的に実行を停止する．これをストールという．これが複数のトランザクションで発生すると，デッドロック状態に陥る可能性がある．そこで，各スレッドが possible_cycle フラグを保持することでこれを回避している．この手法では，あるトランザクションが，より早く開始したトランザクションに対して NACK を返信する際に，自身の possible_cycle フラグをセットする．そして possible_cycle フラグがセットされた状態で，自身よりも早く開始したトランザクションから NACK を受信した場合にトランザクションをアボートする．

3 提案

3.1 アボート条件の厳格化

possible_cycle フラグによるアボート対象選択手法では，アボートが過剰に発生する可能性がある．これは，2 スレッド間における競合情報と，possible_cycle フラグのみを用いてデッドロックを防止するため，実際にはデッドロックを起こしていな場合にもアボートが発生するためである．そこで，トランザクションのアボート条件をデッドロック検出時のみとする手法を提案する．これによりアボートの発生を抑制し，高速化を図る．

3.2 アボート対象の選択

デッドロックを解決する最も簡単な方法は，デッドロックを検出したスレッドが，自身のトランザクションをアボートさせることである．また，デッドロックに関係するトランザクションの中から適切なアボート対象を選択することができれば，さらなる性能向上が期待できる．そこで，アボート対象を選択する二つの手法を提案する．ひとつは，トランザクション開始時

Copyright ©2012 Information Processing Society of Japan. All Rights Reserved.

刻を比較し，より遅く開始したトランザクションをアボート対象とする手法である．これによりスタベーションを防ぐことができる．もう一つは，自身がストールさせているトランザクション数が，より多いトランザクションをアボート対象とする手法である．これにより，多くのトランザクションが再開できると考えられ，並列実行スレッド数の増加が期待できる．

4　実装

4.1　デッドロックの検出

各スレッドは，自身をストールさせているスレッド番号をビット列として記憶しておき，NACK 返信時に併せてそれを送信する．また，NACK 受信時には受信したビット列と自身の保持しているビット列との論理和をとり，さらに相手のスレッド番号に対応するビットをセットする．これによって，自身のスレッド番号のビットがセットされた場合，デッドロックが発生したことを検出する．また，トランザクションをアボート／コミットした際，各スレッドが保持するビット列に矛盾が生じる．そこで，ACK 受信時に競合相手スレッド番号に対応するビットをクリアする．

4.2　アボート対象の選択

提案したアボート対象選択手法を実現するためには，対象の選択に必要な情報を収集する必要がある．これは，自身の持つ情報を NACK とともに伝搬させ，デッドロックを引き起こしている依存関係の輪をもう一度循環させることで実現する．そして，情報収集後にアボート対象となるトランザクションを実行するスレッドへ，アボートリクエストを送信する．

5　評価

5.1　評価結果

評価対象のプログラムとして GEMS 付属ベンチマークプログラムである btree，contention，deque，prioque を用い，それぞれのプログラムを 8，16，31 スレッドで実行した．

図 1 に各プログラムにおける評価結果を示す．凡例はトランザクションにおけるサイクル数の内訳を表しており，non_trans はトランザクション外の実行サイクル数，good_trans はコミットされたトランザクションの実行サイクル数，bad_trans はアボートされたトランザクションの実行サイクル数，aborting はアボートに要したサイクル数，stall はストールに要したサイクル数，backoff はアボート後に実行開始までランダム時間待つサイクル数である．それぞれのグラフは左から順に，(P)possible_cycle フラグを使用する既存手法，(M) デッドロックを検出したスレッドが自身のトランザクションをアボートする手法，(C) 競合数の比較によるアボート対象選択手法，(T) 開始時刻の比較によるアボート対象選択手法 に要した実行サイクル数を表しており，(P) の実行サイクル数を 1 として正規化した．なお，アボート対象選択部分は実装途中であるため，(C)，(T) のサイクル数は情報収集にかかるコストを含んでいない．

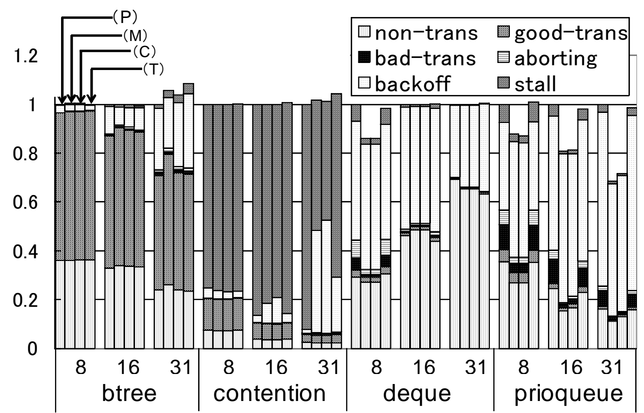

図 1: 各プログラムにおけるサイクル数比

5.2　考察

deque，prioqueue では，(M) によりアボートが抑制され，aborting，bad-trans が削減された．この手法では，31 スレッドで最大 31.5%，平均 6.0%サイクル削減率が向上した．btree ではアボート回数を削減した一方で，命令数の多いトランザクションにアボート対象が偏ったためにログの書き戻しによるコストが増大したと考えられる．また，(C) では並列実行スレッドの増加によりサイクル数が削減されることを，多くのプログラムで確認した．

6　おわりに

複数トランザクション間の競合を考慮した競合解決手法を提案した．今後の課題として，アボート対象選択部分の実装完成が挙げられる．

参考文献

[1] Herlihy, M. and Moss, J. E. B.: Transactional Memory: Architectural Support for Lock-Free Data Structures, *Proc. of 20th Annual International Symposium on Computer Architecture*, ACM, pp. 289–300 (1993).

[2] Moore, K. E. et al.: LogTM: Log-based Transactional Memory, *Proc. of 12th International Symposium on High-Performance Computer Architecture*, IEEE Computer Society, pp. 254–265 (2006).

[3] Censier, L. M. and Feautrier, P.: A New Solution to Coherence Problems in Multicache Systems, *IEEE Transactions on Computers*, Vol. c-27, No. 12, pp. 1112–1118 (1978).

Copyright ©2012 Information Processing Society of Japan.
All Rights Reserved.